Luca Aceto
Monika Henzinger
Jiří Sgall (Eds.)

ARCoSS
LNCS 6756

Automata, Languages and Programming

38th International Colloquium, ICALP 2011
Zurich, Switzerland, July 2011
Proceedings, Part II

2 Part II

Springer

Lecture Notes in Computer Science 6756

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

Advanced Research in Computing and Software Science
Subline of Lectures Notes in Computer Science

Luca Aceto Monika Henzinger
Jiří Sgall (Eds.)

Automata, Languages and Programming

38th International Colloquium, ICALP 2011
Zurich, Switzerland, July 4-8, 2011
Proceedings, Part II

Springer

Volume Editors

Luca Aceto
Reykjavik University, School of Computer Science
101 Reykjavík, Iceland
E-mail: luca@ru.is

Monika Henzinger
Universität Wien, Fakultät für Informatik
Universitätsstraße10/9, 1090 Wien, Österreich
E-mail: monika.henzinger@univie.ac.at

Jiří Sgall
Charles University, Department of Applied Mathematics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
E-mail: sgall@kam.mff.cuni.cz

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-22011-1 e-ISBN 978-3-642-22012-8
DOI 10.1007/978-3-642-22012-8
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011930001

CR Subject Classification (1998): F.3, D.2, F.4, C.2, F.2, H.4

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

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

ICALP 2011, the 38th edition of the International Colloquium on Automata, Languages and Programming, was held in Zürich, Switzerland, during July 4–8, 2011. ICALP is a series of annual conferences of the European Association for Theoretical Computer Science (EATCS) which first took place in 1972. This year, the ICALP program consisted of three tracks: the established Track A (focusing on Algorithms, Complexity and Games) and Track B (focusing on Logic, Semantics, Automata and Theory of Programming), and a Track C focusing on Foundations of Networked Computation: Models, Algorithms and Information Management.

In response to the call for papers, the Program Committee received 398 submissions: 243 for Track A (three of which were later withdrawn), 103 for Track B and 52 for Track C. Out of these, 114 papers were selected for inclusion in the scientific program: 68 papers for Track A, 29 for Track B, and 17 for Track C. The selection was made by the Program Committees based on originality, quality, and relevance to theoretical computer science. The quality of the manuscripts was very high indeed, and many deserving papers could not be selected.

The EATCS sponsored both a best paper and a best student paper (all authors are students) award for each of the three tracks, to be selected by the Program Committees. The best paper awards were given to Malte Beecken, Johannes Mittmann, and Nitin Saxena for their paper "Algebraic Independence and Blackbox Identity Testing" (Track A), to Olivier Carton, Thomas Colcombet, and Gabriele Puppis for their paper "Regular Languages of Words Over Countable Linear Orderings" (Track B), and to Martin Hoefer for his paper "Local Matching Dynamics in Social Networks" (Track C). The best student paper awards were given to Shi Li for his paper "A 1.488-Approximation Algorithm for the Uncapacitated Facility Location Problem" (Track A), to Martin Delacourt for his paper "Rice's Theorem for mu-Limit Sets of Cellular Automata" (Track B), and to Shiri Chechik for her paper "Fault-Tolerant Compact Routing Schemes for General Graphs" (Track C).

ICALP 2011 consisted of five invited lectures and the contributed papers. This volume of the proceedings contains all contributed papers from Track A, except for the two papers that received one of the best paper awards. These two papers are contained in a companion volume, together with all contributed papers from Track B and Track C, and the papers by four of the invited speakers: Rajeev Alur (University of Pennsylvania, USA), Thore Husfeldt (IT University of Copenhagen, Denmark), Catuscia Palamidessi (INRIA Saclay and LIX, France), and Ronen Shaltiel (University of Haifa, Israel). The program had an additional invited lecture by Éva Tardos (Cornell University, USA), which does not appear in the proceedings.

The following workshops were held as satellite events of ICALP 2011:
GA - Graph algorithms and Applications
GT - Group Testing
DCM - 7th International Workshop on Developments of Computational Models
SDKB - 5th International Workshop on Semantics in Data and Knowledge Bases

We wish to thank all the authors who submitted extended abstracts for consideration, the Program Committees for their scholarly effort, and all referees who assisted the Program Committees in the evaluation process.

Thanks to the sponsors (Swiss National Science Foundation and Google) for their support, and to ETH Zürich for hosting ICALP 2011. We are also grateful to all members of the Organizing Committee and to their support staff in the Institute of Theoretical Computer Science at ETH Zürich. The conference-management system EasyChair was used in handling the submissions and the electronic Program Committee meeting, as well as in assisting in the assembly of the proceedings.

May 2011

Luca Aceto
Monika Henzinger
Jiří Sgall

Organization

Program Committee

Track A

Nikhil Bansal	IBM Research, USA
Harry Buhrman	University of Amsterdam, The Netherlands
Marek Chrobak	University of California, Riverside, USA
Martin Dietzfelbinger	Ilmenau University of Technology, Germany
Thomas Erlebach	University of Leicester, UK
Fedor Fomin	University of Bergen, Norway
Dimitris Fotakis	National Technical University of Athens, Greece
Ricard Gavaldà	UPC Barcelona, Spain
Russell Impagliazzo	University of California, San Diego, USA
Juhani Karhumäki	University of Turku, Finland
Howard Karloff	AT&T Labs–Research, USA
Michal Koucký	Academy of Sciences of the Czech Republic, Czech Republic
Dariusz Kowalski	University of Liverpool, UK
Stefano Leonardi	Sapienza University of Rome, Italy
Gonzalo Navarro	University of Chile, Chile
Rolf Niedermeier	Technische Universität Berlin, Germany
Rafail Ostrovsky	University of California, Los Angeles, USA
Günter Rote	Freie Universität Berlin, Germany
Christian Scheideler	University of Paderborn, Germany
Maria Serna	UPC Barcelona, Spain
Jiří Sgall	Charles University in Prague (Chair), Czech Republic
Gábor Tardos	Simon Fraser University, Canada
Jan Vondrák	IBM Research, USA
Uli Wagner	ETH Zürich, Switzerland
Prudence Wong	University of Liverpool, UK

Track B

Luca Aceto	Reykjavik University (Chair), Iceland
Anuj Dawar	University of Cambridge, UK
Rocco De Nicola	University of Florence, Italy
Zoltán Ésik	University of Szeged, Hungary
Wan Fokkink	VU University Amsterdam, The Netherlands
Herman Geuvers	Radboud University Nijmegen, The Netherlands
Radha Jagadeesan	DePaul University, USA

Jarkko Kari	University of Turku, Finland
Joost-Pieter Katoen	RWTH Aachen, Germany
Orna Kupferman	Hebrew University, Israel
Francois Laroussinie	LIAFA, University Paris Diderot, France
Carroll Morgan	University of New South Wales, Australia
Anca Muscholl	LaBRI, University of Bordeaux, France
Hanne Riis Nielson	Technical University of Denmark, Denmark
Prakash Panangaden	McGill University, Canada
Joachim Parrow	Uppsala University, Sweden
Reinhard Pichler	Vienna University of Technology, Austria
Roberto Segala	University of Verona, Italy
Helmut Seidl	Technische Universität München, Germany
Alex Simpson	University of Edinburgh, UK
Pawel Urzyczyn	University of Warsaw, Poland

Track C

Gilles Barthe	IMDEA Software, Spain
András Benczúr	Hungarian Academy of Sciences, Hungary
Edith Cohen	AT&T Labs–Research, USA
Joan Feigenbaum	Yale University, USA
Amos Fiat	Tel-Aviv University, Israel
Lisa Fleischer	Dartmouth College, USA
Georg Gottlob	Oxford University, UK
Monika Henzinger	University of Vienna (Chair), Austria
Bruce Maggs	Carnegie Mellon and Duke University, USA
Massimo Merro	University of Verona, Italy
Vahab Mirrokni	Google Inc., USA
Alessandro Panconesi	Sapienza University of Rome, Italy
Giuseppe Persiano	University of Salerno, Italy
Anna Philippou	University of Cyprus, Cyprus
Davide Sangiorgi	University of Bologna, Italy
Vladimiro Sassone	University of Southampton, UK
Andrew Tomkins	Google Inc., USA
Dorothea Wagner	Karlsruhe Institute of Technology, Germany
Roger Wattenhofer	ETH Zürich, Switzerland
Ingmar Weber	Yahoo! Research Barcelona, Spain

Conference Chairs

Michael Hoffmann	ETH Zürich
Juraj Hromkovič	ETH Zürich
Ueli Maurer	ETH Zürich
Angelika Steger	ETH Zürich
Emo Welzl	ETH Zürich
Peter Widmayer	ETH Zürich

Referees

Farid Ablayev
Lucia Acciai
Pankaj Agarwal
Jae Hyun Ahn
Saeed Alaei
Susanne Albers
Boris Alexeev
Eric Allender
Shaull Almagor
Andris Ambainis
Aris Anagnostopoulos
Alexandr Andoni
Andrea Frosini
Stefan Andrei
Aaron Archer
Argimiro Arratia
Eugene Asarin
James Aspnes
Mohamed Faouzi Atig
Albert Atserias
Jaume Baixeries
Borja Balle
Vince Bárány
Jérémy Barbay
Pablo Barceló
David Mix Barrington
Libor Barto
Mohammadhossein Bateni
Tugkan Batu
Reinhard Bauer
Mohsen Bayati
Paul Beame
Martin Beaudry
Luca Becchetti
Nicolas Bedon
Piotr Berman
Marco Bernardo
Sandford Bessler
Nadja Betzler
René van Bevern
Umang Bhaskar
Binay Bhattacharya
Marcin Bienkowski
Henrik Björklund
Markus Bläser
Ed Blakey
Jens Blanck
Avrim Blum
Hans-Joachim Böckenhauer
Andrej Bogdanov
Paolo Boldi
Francesco Bonchi
Marcello Bonsangue
Johannes Borgström
Glencora Borradaile
Patricia Bouyer
Julian Bradfield
Vasco Brattka
Vladimir Braverman
Robert Bredereck
Davide Bresolin
Franck van Breugel
Patrick Briest
Thomas Brihaye
Gerth Stølting Brodal
Václav Brožek
Janusz Brzozowski
Andrei Bulatov
Sebastian Burckhardt
Sergiu Bursuc
Jaroslaw Byrka
Sergio Cabello
Jin-Yi Cai
Gruia Calinescu
Cristian S. Calude
Philippe Camacho
Silvio Capobianco
Alberto Caprara
Venanzio Capretta
Ioannis Caragiannis
Arnaud Carayol
Olivier Carton
Sourav Chakraborty
Ho-Leung Chan
Joseph Chan
Witold Charatonik
Krishnendu Chatterjee
Arkadev Chattopadhyay
Prasad Chebolu
Shiri Chechik
Panagiotis Cheilaris
Jianer Chen
Jiehua Chen
Joe Cheriyan
Sherman S.M. Chow
Tobias Christ
Giorgios Christodoulou
Richard Cleve
Thomas Colcombet
Florin Constantin
Graham Cormode
José Correa
László Csirmaz
Marek Cygan
Artur Czumaj
Victor Dalmau
Peter Damaschke
Philippe Darondeau
Sanjeeb Dash
Samir Datta
Tugrul Dayar
Brian Dean
Aldric Degorre
Alberto Del Pia
Stephanie Delaune
Holger Dell
Giorgio Delzanno
Yuxin Deng
Josee Desharnais
Mariangiola Dezani
Ilias Diakonikolas
Volker Diekert
Michael Dinitz
Laurent Doyen
Feodor Dragan
Martin Dyer
Stefan Dziembowski

Josep Díaz
György Dósa
Jeff Egger
Pavlos Eirinakis
Christian Eisentraut
Tinaz Ekim
Khaled Elbassioni
Michael Elkin
Yuval Emek
Alina Ene
David Eppstein
Leah Epstein
Guy Even
Rolf Fagerberg
Jean Fanchon
Arash Farzan
Tomás Feder
Ingo Feinerer
Michael Fellows
Henning Fernau
Diodato Ferraioli
Esteban Feuerstein
Piotr Filipiuk
Irene Finocchi
Dario Fiore
Luca Fossati
Pierre Fraignaud
Gianmarco
De Francisci Morales
Rusins Freivalds
Tom Friedetzky
Ehud Friedgut
Hongfei Fu
Takuro Fukunaga
Stanley Fung
Joaquim Gabarro
Murdoch Gabbay
Bernd Gärtner
Martin Gairing
Anna Găl
Nicola Galesi
Pierre Ganty
Leszek Gasieniec
Serge Gaspers
Qi Ge
Heidi Gebauer
Ran Gelles
Blaise Genest
Chryssis Georgiou
George Giakkoupis
Panos Giannopoulos
Hugo Gimbert
Aristides Gionis
Vasilis Gkatzelis
Rob van Glabbeek
Andreas-Nikolas Gobel
Andreas Goerdt
Leslie Ann Goldberg
Paul Goldberg
Oded Goldreich
Mordecai J. Golin
Petr Golovach
Renato Gomes
Gosta Grahne
Fabrizio Grandoni
Vince Grolmusz
Jan Friso Groote
Romain Grunert
Erich Grädel
Sudipto Guha
Oktay Gunluk
Jiong Guo
Dan Gutfreund
Gregory Gutin
Robert Görke
Annegret Habel
Serge Haddad
Mohammadtaghi
Hajiaghayi
Magnús M. Halldórsson
Sean Hallgren
Nir Halman
Sardouna Hamadou
Xin Han
Chris Hankin
Kristoffer
Arnsfelt Hansen
Nicolas Hanusse
Sariel Har-Peled
Tero Harju
Tobias Harks
Sepp Hartung
Ichiro Hasuo
Pinar Heggernes
Brett Hemenway
Matthew Hennessy
Alejandro Hernandez
Timon Hertli
Jane Hillston
Edward A. Hirsch
Mika Hirvensalo
John Hitchcock
Petr Hliněný
Martin Hoefer
Frank Hoffmann
Thomas Holenstein
Lukas Holik
Wing Kai Hon
Iiro Honkala
Hendrik Jan Hoogeboom
Juraj Hromkovič
Pierre Hyvernat
Falk Hüffner
Martina Hüllmann
Nicole Immorlica
Yuval Ishai
Giuseppe F. Italiano
Szabolcs Iván
Kazuo Iwama
Andreas Jakoby
David N. Jansen
Klaus Jansen
Emmanuel Jeandel
Mark Jerrum
Ranjit Jhala
Albert Xin Jiang
David Johnson
Peter Jonsson
Hossein Jowhari
David Juedes
Joanna Jędrzejowicz
Valentine Kabanets
Mark Kambites
Marcin Kamiński
Ohad Kammar

Frank Kammer
Mong-Jen Kao
Alexis Kaporis
Michael Kapralov
George Karakostas
David Karger
Lila Kari
Juha Karkkainen
Dmitriy Katz
Telikepalli Kavitha
Ken-Ichi Kawarabayashi
Bart de Keijzer
Alexander Kesselman
Andrew King
Daniel Kirsten
Tamás Király
Hartmut Klauck
Philip Klein
Marek Klonowski
Christian Knauer
Sebastian Kniesburges
Naoki Kobayashi
Johannes Köbler
Jochen Könemann
Boris Köpf
Pascal Koiran
Stavros Kolliopoulos
Petr Kolman
Christian Komusiewicz
Stavros Konstantinidis
Spyros Kontogiannis
Eryk Kopczynski
Guy Kortsarz
Nitish Korula
Adrian Kosowski
Paraschos Koutris
Andreas Koutsopoulos
Łukasz Kowalik
Miroslaw Kowaluk
Marcin Kozik
Jan Krajíček
Dieter Kratsch
Robbert Krebbers
Klaus Kriegel
Ravishankar Krishnaswamy
Andrei Krokhin
Marcus Krug
Piotr Krysta
Manfred Kudlek
Fabian Kuhn
Oliver Kullmann
Ravi Kumar
Gábor Kun
Alexander Kurz
Martin Kutrib
Tomi Kärki
Juha Kärkkäinen
Oded Lachish
Pascal Lafourcade
Tak-Wah Lam
Michael Lampis
Lawrence Larmore
Kasper Green Larsen
Sławomir Lasota
Diego Latella
Silvio Lattanzi
Lap Chi Lau
Luigi Laura
Lap-Kei Lee
Troy Lee
Erik Jan van Leeuwen
Pietro Di Lena
Jérôme Leroux
Adam Letchford
Peter Leupold
Jian Li
Ugo de'Liguoro
Andrzej Lingas
Ben Lippmeier
Jiamou Liu
Ivana Ljubic
Martin Loebl
Bruno Loff
Markus Lohrey
Daniel Lokshtanov
Sylvain Lombardy
Violetta Lonati
John Longley
Michele Loreti
Antoni Lozano
Alessandro De Luca
Edward Lui
Christof Löding
Benedikt Löwe
Christoph Lüth
Klaus Madlener
Aleksander Madry
Mohammad Mahdian
Hemanta Maji
Konstantin Makarychev
Yury Makarychev
David F. Manlove
Rajsekar Manokaran
Giulio Manzonetto
Martin Mareš
Nicolas Markey
Bruno Marnette
Barnaby Martin
Russell Martin
Conrado Martinez
Dániel Marx
Mieke Massink
Monaldo Mastrolilli
Claire Mathieu
Arie Matsliah
Elvira Mayordomo
Andrew McGregor
Annabelle McIver
Pierre McKenzie
Larissa Meinicke
Daniel Meister
Páll Melsted
Wolfgang Merkle
George B. Mertzios
Stephan Merz
Julián Mestre
Roland Meyer
Tillmann Miltzow
Matteo Mio
Neeldhara Misra
Michael Mitzenmacher

Xavier Molinero
Ashley Montanaro
Georg Moser
Elchanan Mossel
Achour Mostefaoui
David Mount
Haiko Müller
Wolfgang Mulzer
Marcelo Mydlarz
Sebastian A Mödersheim
Rasmus Ejlers Møgelberg
Viswanath Nagarajan
Mark-Jan Nederhof
Alantha Newman
Ilan Newman
Cyril Nicaud
André Nichterlein
Dejan Nickovic
Flemming Nielson
Damian Niwinski
Martin Nöllenburg
Thomas Noll
Gethin Norman
Dirk Nowotka
Zeev Nutov
Jan Obdržálek
Alexander Okhotin
Sergi Oliva
Alexander Olshevsky
Krzysztof Onak
Adrian Onet
Luke Ong
Rotem Oshman
Friedrich Otto
Shayan Oveis Gharan
Scott Owens
Rasmus Pagh
Thomas Pajor
Katarzyna Paluch
Konstantinos Panagiotou
Debmalya Panigrahi
Charis Papadopoulos
Vicky Papadopoulou
Srinivasan Parthasarathy
Pawel Parys
Francesco Pasquale
Balázs Patkós
Sriram Pemmaraju
Holger Petersen
Ion Petre
Cynthia Phillips
Claudine Picaronny
George Pierrakos
Giovanni Pighizzini
Marcin Pilipczuk
Adolfo Piperno
Giuseppe Pirillo
Nir Piterman
Wojciech Plandowski
Jaco van de Pol
C.K. Poon
Alexandru Popa
Viorel Preoteasa
Christian W. Probst
Kirk Pruhs
Rosario Pugliese
Dömötör Pálvölgyi
Karin Quaas
Jose Quaresma
Femke van Raamsdonk
Alexander Rabinovich
Yuri Rabinovich
Luis Rademacher
Tomasz Radzik
Vijaya Ramachandran
Rajeev Raman
Venkatesh Raman
Narad Rampersad
Julian Rathke
Dror Rawitz
Ran Raz
Igor Razgon
Oded Regev
Klaus Reinhardt
Tzachy Reinman
Michel Reniers
Pierre-Alain Reynier
Mark Reynolds
Michael Rink
Liam Roditty
Heiko Röglin
Dana Ron
Mads Rosendahl
Alexander Russell
Ignaz Rutter
Andrey Rybalchenko
Wojciech Rytter
Harald Räcke
Stefan Rümmele
Joshua Sack
Mert Saglam
Jacques Sakarovitch
Mohammad Salavatipour
Kai Salomaa
Alex Samorodnitsky
Peter Sanders
Miklos Santha
Rahul Santhanam
Rik Sarkar
Saket Saurabh
Vadim Savenkov
Nitin Saxena
Chrstian Schaffner
Dominik Scheder
Marc Scherfenberg
Saul Schleimer
Lena Schlipf
Philippe Schnoebelen
Aleksy Schubert
Warren Schudy
Daria Schymura
Géraud Sénizergues
Marco Serafini
Olivier Serre
Rocco Servedio
C. Seshadhri
Mordechai Shalom
Arpit Sharma
Sarai Sheinvald
Akiyoshi Shioura
David Shmoys
Igor Shparlinski
Amir Shpilka
Anastasios Sidiropoulos

Mihaela Sighireanu
Pedro V. Silva
Riccardo Silvestri
Hans Ulrich Simon
Sebastian Skritek
Nataliya Skrypnyuk
Martin Skutella
Michael Smith
Christian Sohler
Ana Sokolova
Manuel Sorge
Paul Spirakis
Jeremy Sproston
Jiří Srba
Kannan Srinathan
Juraj Stacho
Julinda Stefa
Daniel Štefankovič
Fabian Stehn
Colin Stirling
Alejandro Strejilevich de Loma
David Steurer
Kristian Støvring
Ondřej Suchý
Ola Svensson
Maxim Sviridenko
Chaitanya Swamy
Vasilis Syrgkanis
Stefan Szeider
Nicolas Tabareau
Kunal Talwar
Till Tantau
Éva Tardos
Nina Taslaman
Orestis Telelis
Balder Ten Cate
Lidia Tendera
Michał Terepeta
Pascal Tesson
Nithum Thain
Dimitrios Thilikos
Mikkel Thorup
Francesco Tiezzi
Hingfung Ting
Ashish Tiwari
Szymon Toruńczyk
Mirco Tribastone
Stavros Tripakis
Rahul Tripathi
Enrico Tronci
Madhur Tulsiani
Christos Tzamos
Nikos Tzevelekos
Géza Tóth
Ryuhei Uehara
Marc Uetz
Johannes Uhlmann
Irek Ulidowski
Ugo Vaccaro
Sándor Vágvölgyi
Vasco T. Vasconcelos
Andrea Vattani
Roope Vehkalahti
Helmut Veith
Santosh S. Vempala
Betti Venneri
Carmine Ventre
Elad Verbin
Oleg Verbitsky
Nikolay Vereshchagin
Ivan Visconti
Mahesh Viswanathan
Berthold Vöcking
Markus Völker
Walter Vogler
Björn Wachter
Lei Wang
John Watrous
Mathias Weller
Daniel Werner
Matthias Westermann
Andreas Wiese
Thomas Wilke
Gerhard J. Woeginger
Ronald de Wolf
Paul Wollan
Frank Wolter
David Woodruff
James Worrell
Yi Wu
Dachuan Xu
Eran Yahav
Li Yan
Qiqi Yan
Mu Yang
Ke Yi
Neal Young
Raphael Yuster
Morteza Zadimoghaddam
Michael Zakharyaschev
Hans Zantema
Christos Zaroliagis
Konrad Zdanowski
Marc Zeitoun
Alex Zelikovsky
Rico Zenklusen
Fuyuan Zhang
Lijun Zhang
Min Zhang
Yong Zhang
Chunlai Zhou
Stanislav Živný
Florian Zuleger

Table of Contents – Part II

Invited Lectures

Best Student Papers

Best Papers

Session B1: Foundations of Program Semantics

Session B2: Automata and Formal Languages

Session B3: Model Checking

Session B4: Probabilistic Systems

Session B5: Logic in Computer Science

Session B6: Hybrid Systems

Session B7: Specification and Verification

Session C1: Graphs

Session C2: Matchings and Equilibria

Session C3: Privacy and Content Search

Session C4: Distributed Computation

Table of Contents – Part I

Session A1: Network Design Problems

Session A2: Quantum Computing

Session A3: Graph Algorithms

Session A4: Games, Approximation Schemes, Smoothed Analysis

Session A5: Online Algorithms

Session A6: Data Structures, Distributed Computing

Session A7: Complexity, Randomness

Session A8: Submodular Optimization, Matroids

Session A9: Cryptography, Learning

Session A10: Fixed Parameter Tractability

Session A11: Hardness of Approximation

Session A12: Counting, Testing

Session A13: Complexity

Session A14: Proof Complexity

Session A15: Sorting, Matchings, Paths

Session A16: Constraint Satisfaction, Algebraic Complexity

Session A17: Steiner Problems, Clustering

Nondeterministic Streaming String Transducers*

Rajeev Alur and Jyotirmoy V. Deshmukh

Dept. of Computer and Information Science,
University of Pennsylvania
{alur,djy}@cis.upenn.edu

Abstract. We introduce *nondeterministic streaming string transducers* (NSSTs) – a new computational model that can implement MSO-definable relations between strings. An NSST makes a single left-to-right pass on the input string and uses a finite set of *string variables* to compute the output. In each step, it reads one input symbol, and updates its string variables in parallel with a *copyless assignment*. We show that NSST are closed under sequential composition and that their expressive power coincides with that of nondeterministic MSO-definable transductions. Further, we identify the class of *functional* NSSTs; such an NSST allows nondeterministic transitions, but for every successful run on a given input generates the same output string. We show that deciding functionality of an arbitrary NSST is decidable with PSPACE complexity, while the equivalence problem for functional NSSTs is PSPACE-COMPLETE. We also show that checking if the set of outputs of an NSST is contained within the set of outputs of a finite number of DSSTs is decidable in PSPACE.

1 Introduction

In this paper, we introduce *nondeterministic streaming string transducers* (NSSTs). A run of such a transducer processes an input string in a single left-to-right pass in linear time, and computes an output string. The nondeterminism of an NSST allows it to produce different outputs for different runs on an input string. Thus, an NSST is a natural model for implementing *relations* between strings.

Classical literature on string-to-string transductions largely focusses on finite-state transducers that realize *rational relations* [12]. In each step, such a transducer reads an input symbol and writes zero or more symbols to the output. These transducers and their many variations have been extensively studied in the literature. If restricted to a single left-to-right pass over the input, *i.e.*, to their streaming versions, the expressive power of finite transducers is limited. For example, they can implement transductions such as inserting a symbol in an input string and deleting a substring of the input string, but they cannot implement transductions such as reversing an input string or swapping two substrings within an input string.

* This research is partially supported by NSF award CCF 0905464 and by the CCC-CRA Computing Innovation Fellows project.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 1–20, 2011.

Deterministic streaming string transducers (DSSTs), first introduced in [1,2], in addition to all transductions implemented by deterministic finite transducers, can implement transductions such as reversing a string and swapping substrings. A DSST reads the input in a single left-to-right pass. In addition to a finite set of states, it has a finite set of string variables that it uses to produce the output. In each step, a DSST reads an input symbol, changes its state, and concurrently updates all its string variables using a *copyless assignment*. The right-hand-sides (RHS) in a copyless assignment consist of a concatenation of string variables and output symbols, with the restriction that in a parallel assignment, a variable can appear at most once across all right-hand-sides.

A DSST that reverses a string uses a single state and one string variable x. In each step, if it reads the symbol a, it executes the assignment $[x := a.x]$ (here '.' is the string concatenation operator). At the end of its computation, x contains the string that is the reverse of the input string. As another example, consider the transformation mapping a string of the form l,m_f to the string f_m_l (here l, m and f respectively denote strings corresponding to some last, middle, and first name, and '_' denotes a single space). A DSST with two variables x and y can implement this transduction. It stores the substring till the comma (*i.e.*, l) in x, and stores the subsequent string (m) till it sees the space in y. It then sets y to $y.\_.x$, resets x to the empty string, and stores the remaining substring (f) in x. It finally outputs the string $x.\_.y$.

Compared to their deterministic counterparts that implement partial functions between strings, nondeterministic finite transducers (NFTs) can implement relations between strings; for example, mapping a string to all its substrings. However, NFTs lack the expressiveness to implement transductions such as (1) mapping a string w to all strings w' obtained by swapping some prefix and suffix of w, (2) mapping w to all strings $w' = p.rev(m).s$, where w has the form $p.m.s$ (where p, m and s are substrings of w), and $rev(m)$ denotes the reverse of m[1].

In addition to all transductions realized by NFTs, NSSTs can implement both of these transductions. The first transduction is implemented by an NSST with string variables x and y. It nondeterministically chooses some prefix p to store in x and some following substring m to store in y. It then sets y to $y.x$, resets x, copies the remaining input s to x, and finally outputs $x.y$. The NSST implementing the second transduction copies some prefix p (chosen nondeterministically) to x, and uses y to compute $rev(m)$ for a nondeterministically chosen m. It then sets x to $x.y$, appends the remaining input s to x, and finally outputs x.

The organization of the paper is as follows: After defining the NSST model in Sec. 2, we characterize the expressive power of NSSTs in Sec. 3. We show that NSSTs are closed under sequential composition, and prove that the expressive power of NSSTs is equivalent to that of nondeterministic MSO-definable transductions. We then compare both NSSTs and ε-NSSTs– an extended model that allows ε-transitions – with classical models such as two-way nondeterminis-

[1] For a given prefix p and a given suffix s, the transformation from $p.m.s$ to $p.rev(m).s$ is well-known as the *inversion operator* for a string representation of a chromosome [10]. Thus, this transduction generates all inversions of a string.

tic generalized sequential machines. In Sec. 4, we explore decision problems for NSSTs and their subclasses. An interesting and robust subclass of NSSTs is that of *functional* NSSTs: these can have multiple runs on a given input string, but the output produced in each run is identical. We show that checking whether an NSST is functional is decidable in PSPACE, and show that checking equivalence of functional NSSTs is PSPACE-COMPLETE.

In [2], the authors show how DSSTs can be viewed as natural models for a class of *sequential* heap-manipulating programs, and reduce the verification problem for such programs to decidable problems for DSSTs. A key application for NSSTs would be in understanding the theoretical limits of verification problems for *concurrent* heap-manipulating programs. Here, we wish to verify if the set of behaviors of the concurrent program (say $m_1 \parallel m_2$) is a subset of the set of behaviors engendered by some sequential execution of the programs m_1 and m_2. For instance, checking *linearizability* of a set of methods often entails such a check. With this goal in mind, we show that checking if the set of outputs of an NSST is contained within the set of outputs generated by a fixed number of DSSTs is decidable with PSPACE complexity.

2 Transducer Models

In this section, we introduce definitions related to transductions, review existing models for specifying transductions, and formally define NSSTs. We observe the following convention: the letters R and W denote transductions, and the letters T, D denote the transducers implementing transductions.

Transduction. A transduction R from a finite input alphabet Σ to a finite output alphabet Γ is a subset of $\Sigma^* \times \Gamma^*$. A transduction is *deterministic* if for every $x \in \Sigma^*$, there is at most one y such that $(x, y) \in R$, *i.e.*, R is a partial function. Otherwise, we say that the transduction is *nondeterministic*. Given an input $x \in \Sigma^*$, we use $R(x)$ to denote the set $\{y \mid (x, y) \in R\}$.

Valuedness. A transduction is called *finitary* if for every $x \in \Sigma^*$, the set $R(x)$ is finite. A transduction R is said to be *k-valued* for a given constant k, if the following holds: $\forall x \in \Sigma^* : |R(x)| \leq k$. A transduction is said to be bounded-valued if there *exists* some constant k such that R is k-valued. A 1-valued transduction is also called a *functional* transduction. A transduction is called *bounded-length* if there exists a constant k such that for all $x \in \Sigma^*$ and for each y in $R(x)$, $|y| \leq k.|x|$. Note that a bounded-valued transduction is obviously finitary, while every bounded-length transduction is also finitary: for a constant k, and a given input x, there are finitely many strings of length $k.|x|$.

Example 2.1. Consider the transduction $R_{cp} \subseteq \{a\}^* \times (\Gamma \cup \{\#\})^*$ defined as the set $\{(a^n, w\#w) | n \geq 0, w \in \Gamma^*, |w| = n\}$. R_{cp} maps an input a^n to a string containing two identical copies of some output string of length n, separated by #. As there are only finitely many strings of length n, R_{cp} is finitary. Further, note that R_{cp} is bounded-length: for an input of length n, the output length is

$(2.n + 1)$. However, R_{cp} is not bounded-valued as the number of output strings for an input of length n is $|\Gamma|^{2n}$.

Example 2.2. Consider the transduction $R_{mult} \subseteq \Sigma^* \times \Sigma^*$ defined as the set $\{(w, w^m) \mid m \geq 1, w \in \Sigma^*\}$. R_{mult} is nonfinitary, as it maps the input string w to the set of *all* multiples of the string w. Clearly, R_{mult} is neither bounded-valued nor bounded-length.

2.1 Background

We assume that each machine in the following discussion has finite input and output alphabets Σ and Γ respectively. Each machine reads words from Σ^*, and outputs words in Γ^*. We use the notation $[\![M]\!]$ to define the semantics of M: for an input string $w \in \Sigma^*$, $[\![M]\!](w)$ defines the set of outputs of M on input w.

Finite Transducers. A *finite sequential transducer* (denoted as NFT) is a finite-state device that has a one-way read-only input tape and a one-way output tape. It scans the input tape from the left to the right, and in each state it reads an input symbol, writes a finite string to the output tape, changes its state, and moves its reading head to the next position on the input tape. In each such step, it can nondeterministically choose its next state and the output string that it writes. The output of an NFT is the string on the output tape if the NFT finishes scanning the input tape in some designated final state. Formally, we define an NFT T as the tuple $(Q, \Sigma, \Gamma, E, q_0, F)$ where Q is a finite nonempty set of states, Σ and Γ are input and output alphabets, E is a set of transitions defined as a finite subset of $Q \times (\Sigma \cup \{\varepsilon\}) \times Q \times \Gamma^*$, the state $q_0 \in Q$ is an initial state, and $F \subseteq Q$ is a set of final states. An NFT may contain transitions in which the NFT writes some output string or changes state without moving its reading head. We call such transitions as ε-transitions, and a NFT in which such moves are disallowed is called a ε-free NFT or a *nondeterministic generalized sequential machine* (NGSM).

Two-way Machines. A *two-way nondeterministic generalized sequential machine* (denoted 2NGSM) is a finite-state device with a two-way read-only input tape and a one-way output tape. In each step, the machine reads an input symbol, changes its state, writes a finite string to its output tape, and moves its input head as per its finite-state control. The input head either does not move (denoted by 0), or is moved to the left $(-)$ or to the right $(+)$. In each such move, it has a finite number of nondeterministic choices for the next state, output string written and the direction for moving its head. The string on the input tape is assumed to be $\vdash w \dashv$, where $\vdash$ and $\dashv$ $(\notin \Sigma)$ are special symbols known as the left and right end-markers. The NGSM halts if it moves right from $\dashv$, or left from $\vdash$. It is possible that the computation of the machine may not terminate; however, only halting runs contribute to the output. The output of the machine is the string on the output tape if the machine terminates in a designated final state. Formally, a 2NGSM is specified as the tuple $(Q, \Sigma, \Gamma, E, q_0, F)$, where Q, Σ, Γ, q_0 and F are defined as for finite transducers, while the set of transitions E is defined as a finite subset of $(Q \times \Sigma \cup \{\vdash, \dashv\} \times Q \times \{-, 0, +\} \times \Gamma^*)$.

Determinism and Valuedness. The definition of determinism is the standard one: no two distinct transitions starting in a state q have the same input symbol symbol a. The deterministic versions of the machines defined above are denoted as DFT, DGSM, and 2DGSM. The transductions implemented by a DFT, a DGSM and a 2DGSM are obviously 1-valued (and thus finitary). Further, these transductions can also be shown to be bounded-length, as on an input of length n, these machines take at most n steps, and each step contributes a constant number of symbols to the output. In an NGSM, there are only finitely many paths corresponding to each input string, and the length of output along each such path grows proportional to that of the input. Thus, the transductions implemented by an NGSM are bounded-length (and thus finitary) as well. However, note that the transductions implemented by an NGSM are not bounded-valued in general. The transductions implemented by NFTs and 2NGSMs are neither finitary nor bounded-length.

MSO-definable String Transductions. String transductions can be described using **M**onadic **S**econd **O**rder logic [4,6]. The input to such a transduction is a string $w = w_1 w_2 \dots w_k$ viewed as an input string graph G_i with $k+1$ vertices $v_0, \dots, v_k$, where the label of each edge (v_i, v_{i+1}) is w_i. The labels for the vertices and the edges appearing in the output string graph are expressed in terms of MSO formulas over a fixed number of copies of G_i. Formally, a deterministic MSO-definable string transduction (DMSOS) W is specified by[2] : (1) a finite *copy set* C, (2) for each $c \in C$, vertex formulas $\varphi^c(x)$, which are MSO formulas with one free first-order variable variable x, and (3) for each $a \in (\Gamma \cup \{\varepsilon\})$, $c, d \in C$, edge formulas $\varphi_a^{c,d}(x, y)$, which are MSO formulas with two free first order variables x and y. The output graph (G_o) is defined as follows: for each vertex x in G_i and $c \in C$, the vertex x^c is in G_o if the vertex formula $\varphi^c(x)$ is *true*, and for all vertices x^c, y^d, there is an edge (x^c, y^d) labeled by a if the edge formula $\varphi_a^{c,d}(x)$ is *true*. Finally, the output is defined as the string formed by the sequence of symbols (other than ε) that are the obtained by reading the labels of edges of the output graph G_o in order. If G_o is not a legal string graph the output is undefined. In case of MSO-definable transductions, we use the notation $[\![W]\!](x)$ synonymously with $W(x)$, where x is an input string.

The nondeterministic variant of MSO-definable string transductions is obtained by adding free second-order variables $\mathcal{X}_1, \dots, \mathcal{X}_n$ that range over sets of vertices. The vertex and edge formulas in this case have these variables as additional arguments. To obtain the output string graph, a (global) valuation of the second-order variables $\mathcal{X}_1, \dots, \mathcal{X}_k$ is chosen, and then the output string graph is obtained as in the deterministic case. As the valuation for $\mathcal{X}_1, \dots, \mathcal{X}_k$ is chosen nondeterministically, there can be multiple possible outputs for a given input string.

[2] We present a definition slightly modified from the one in [6]. For ease of exposition, we omit specification of a domain formula φ_{dom}, as it can be included to be a part of the vertex and edge formulas. We also allow edges in the output graph to be labeled with the empty string ε. It can be shown that such MSO-transductions are equivalent to MSO-transductions that do not make use of ε-labels.

Example 2.3. Consider the transduction $R_{ss} = \{(w, u\#v) | u, v$ are subsequences of $w\}$, where $u, v, w \in \Sigma^*$, and $\# \notin \Sigma$. We define R_{ss} as a NMSOS transduction W by choosing the copy set $C = \{1, 2\}$, and two parameters $\mathcal{X}_1$ and $\mathcal{X}_2$. We use the formula $\mathsf{edge}_a(x, y)$ to denote that there is an edge labeled a between vertices x and y, and the formula $\mathsf{last}(x)$ (resp. $\mathsf{first}(x)$) to denote that x is the last (resp. first) node in the input string graph. We now define the edge formulas: $\varphi_{\#}^{1,2}(x, y, \mathcal{X}_1, \mathcal{X}_2) \equiv \mathsf{last}(x) \wedge \mathsf{first}(y)$, and, $\forall j \in C, \forall a \in \Sigma$: $\varphi_a^{j,j}(x, y, \mathcal{X}_1, \mathcal{X}_2) \equiv (x \in \mathcal{X}_j) \wedge \mathsf{edge}_a(x, y)$, and $\forall j \in C$: $\varphi_\varepsilon^{j,j}(x, y, \mathcal{X}_1, \mathcal{X}_2) \equiv (x \notin \mathcal{X}_j) \wedge \bigvee_{a \in \Sigma} \mathsf{edge}_a(x, y)$. Intuitively, the second order variable $\mathcal{X}_1$ is used to guess which positions of the original string contribute to u, while $\mathcal{X}_2$ independently guesses which positions contribute to v. □

Hennie Machines. A two-way machine with a writable input tape and a write-only output tape is defined in similar fashion to a 2NGSM. The instructions of such a machine M are of the form $(q, \sigma/\alpha, q', \gamma)$, which means that in state q, M reads the symbol σ from the tape, overwrites it with the symbol α, transitions to the state q' and writes the finite string γ to the output tape. Here, $\sigma, \alpha \in \Sigma$, where Σ is the tape alphabet, and $\gamma \in \Gamma^*$, which is the output alphabet. A computation of M is called k-visiting, if each position of the tape is visited *at most* k times. A *Hennie machine* is such a two-way machine with the property that there exists a constant k such that every computation of the machine is k-visiting. The classes of string transductions realized by nondeterministic and deterministic Hennie machines are denoted by NHM and DHM, respectively.

Example 2.4. Consider the transduction R_{cp} of Ex. 2.1. In [6], it is shown that R_{cp} can be realized by an NHM moving in two left-to-right passes over the tape. In the first pass, it nondeterministically overwrites the input string a^n with some output string w, and also writes w to the output. It then scans backward to the beginning of the (overwritten) input, writes # to the output, and copies the contents of the input tape to the output a second time. The NHM is 3-visit. □

2.2 Streaming String Transducers

Deterministic streaming string transducers (DSST) have been proposed as an effective machine model for MSO-definable string transductions [2,1]. A DSST makes a *single* left-to-right pass over the input string mapping it to the output string, while using a fixed set of string variables that can store strings over the output alphabet. In each step, a DSST reads an input symbol a, changes its state and updates all its string variables simultaneously using a *copyless* assignment. The right-hand-side (RHS) of any assignment to a string variable is a concatenation of output symbols and some string variables, with the restriction that in a given parallel assignment, any string variable can appear at most once across all the right-hand-sides. For instance, let $X = \{x, y\}$ be the set of string variables, and let $\alpha, \beta, \gamma \in \Gamma^*$ be strings of output symbols. Then, the update $(x, y) = (\alpha.x.\beta.y.\gamma,\ \varepsilon)$ is a copyless assignment, as it contains only

one occurrence of x and y each. On the other hand, the assignment $(x, y) = (x.y,\ y.\alpha)$ is not copyless as the variable y appears in the RHS twice. The two main extensions to the DGSM model are that (1) DSSTs are not constrained to add output symbols only at the end of the tape, and (2) they can compute the output in multiple chunks that can be extended and concatenated (but not duplicated) as needed.

We now present a nondeterministic extension of DSSTs: the class of *nondeterministic streaming string transducers* (NSSTs). An NSST is defined as the tuple $(Q, \Sigma, \Gamma, X, E, q_0, F)$, where Q is a finite nonempty set of states, Σ and Γ are respectively the input and output alphabets, X is a finite set of *string variables* with A as a set of *copyless assignments* to variables in X (mappings α from X to $(X \cup \Gamma)^*$ such that for any $x \in X$, x appears at most once in the set $\{\alpha(y) \mid y \in X\}$), E is a set of transitions that is a finite subset of $(Q \times \Sigma \times A \times Q)$, q_0 is an initial state, and $F : Q \to (X \cup \Gamma)^*$ is a partial output function such that for every $q \in Q$ and $x \in X$, there is at most one occurrence of x in $F(q)$.

Semantics. We first make an observation: copyless assignments are closed under sequential composition, *i.e.*, if α_1 and α_2 are copyless assignments, the assignment $\alpha_1 \circ \alpha_2$ is also copyless. We define the semantics of an NSST in terms of the *summary* of a computation of the NSST. For an NSST starting in state q and processing the input string w, the summary is a set of pairs of the form (α, q'), where α represents the effect of a sequence of copyless assignments to the string variables, and q' is a possible next state. Let id denote the identity assignment, *i.e.*, it maps each x to x. We inductively define the summary Δ as a mapping from $Q \times \Sigma^*$ to $2^{A \times Q}$, where: $\Delta(q, \varepsilon) = \{(\mathsf{id}, q)\}$, and $\Delta(q, w.a) = \{(\alpha_w \circ \alpha, q') \mid \exists q_1, \text{ s.t. } (\alpha_w, q_1) \in \Delta(q, w) \ \wedge \ (q_1, a, \alpha, q') \in E\}$.

A valuation of a string variable is defined as the function $\nu : X \to \Gamma^*$, which maps each variable to some string in Γ^*. Such a valuation is extended to map strings in $(X \cup \Gamma)^*$ to Γ^* in a natural fashion. Let ν_ε be the initial valuation, where for all $x \in X$, $\nu_\varepsilon(x) = \varepsilon$. We define the semantics of an NSST T as the set of outputs $[\![T]\!](w)$ generated by T on input string w:

$$[\![T]\!](w) = \{\nu_\varepsilon(\alpha_w \circ F(q)) \mid (\alpha_w, q) \in \Delta(q_0, w) \text{ and } F(q) \text{ is defined}\}.$$

We now illustrate the model using a couple of examples.

Example 2.5. The NSST T_{ss} implementing the transduction R_{ss} from Ex. 2.3 has a single state q and uses two string variables x and y. For each input symbol a, it nondeterministically decides to append a to x or y or both or none. Formally, the transitions of T_{ss} are of the form $(q, a, (x, y) := (x.\gamma_1, y.\gamma_2), q)$, where $\gamma_1, \gamma_2 \in \{\varepsilon, a\}$. The output function is defined as $F(q) = x.\#.y$. □

Example 2.6. The NSST T_{cp} implementing the transduction R_{cp} from Ex. 2.1 has a single state q and uses two string variables x and y. For each input symbol it nondeterministically chooses an output symbol $\gamma \in \{a, b\}$ and appends it to both x and y, *i.e.*, the set of transitions $E = \{(q, a, (x, y) := (x.\gamma, y.\gamma), q) \mid \gamma \in \{a, b\}\}$. The output function is defined as $F(q) = x.\#.y$. □

Extensions and Subclasses. An NSST with ε-transitions (ε-NSST) can update its string variables and change state without consuming an input symbol, *i.e.* E is a finite subset of $(Q \times (\Sigma \cup \{\varepsilon\}) \times A \times Q)$. As we will see in Sec. 3, an ε-NSST is more powerful than an NSST as it can implement nonfinitary transductions.

Lastly, we note some important subclasses of NSSTs, based on their valuedness. An NSST T is said to be *k-valued*, if the underlying transduction that it implements is k-valued, *i.e.* for a given constant k, $\forall w \in \Sigma^*$, $|[\![T]\!](w)| \leq k$. If there exists some constant k such that T is k-valued, we say that T is *bounded-valued*, and if $k = 1$, we say that the NSST is *functional*. In other words, all runs of a functional NSST on the same input string, which end in a state q where $F(q)$ is defined, produce the same output string. Thus, the transduction realized by such an NSST is a partial function. Functional NSSTs are an interesting subclass of NSSTs, as their decision problems have good algorithmic properties.

3 Expressiveness Results

In this section, we first show that NSSTs are closed under sequential composition, and they implement bounded-length transductions. We then compare NSSTs with the class of NMSOS-transductions, and show that they have the same expressive power. Lastly, we compare the expressive power of ε-NSSTs with that of 2NGSMs.

3.1 Properties of NSSTs

Sequential Composition of DSSTs. Let (Σ, Γ)-DSST be the shorthand for a DSST that has Σ and Γ as its input and output alphabets respectively. Let T_1 be a (Σ_1, Σ_2)-DSST, with the set of states Q_1, and the set of string variables X_1, and let T_2 be a (Σ_2, Σ_3)-DSST with the set of states Q_2 and the set of string variables X_2. We review the construction from [1] to construct the (Σ_1, Σ_3)-DSST $T_s = T_1 \circ T_2$, *i.e.*, T_s is the sequential composition of T_1 and T_2. Let the set of states and the set of string variables of T_s be Q_s and X_s respectively. When computing the sequential composition, the input to T_2 is the output of T_1. Hence, the contents of any string variable $x \in X_1$ are a possible input to T_2. T_s encodes the sequential composition as follows: (1) it simulates the action of T_1 by recording the state of T_1 in its state, and, (2) at each step during the simulation, it records maps f and g in its state, and uses these in conjunction with its string variables to record the summary of T_2 starting in each possible state $r \in Q_2$, on the contents of each string variable $x \in X_1$.

We now formally define these maps. Recall that $\nu(x)$ denotes the valuation of the variable x. We define the map f from $Q_2 \times X_1$ to Q_2, such that $f(r, x) = r'$ if $\Delta_2(r, \nu(x)) = (\alpha', r')$. For each state $r \in Q_2$ and each string variable $x \in X_1$, T_s records the map $f(r, x)$ in its state. A copyless assignment α in T_2 maps each $y \in X_2$ to some concatenation of symbols from Σ_3 (the output language of T_2) and X_2. Thus the *shape* of α is some string of the form $s_1.y_1.s_2.y_2.s_3$, where the s_k's are some finite strings in Σ_3^* and y_j's are variables in X_2. T_s uses its string variables $z \in X_s$ to store these s_k chunks in such a manner that (1) no

two variables in X_s appear consecutively, and (2) none of the variables in X_s or X_2 are repeated (in keeping with the copyless restriction on assignments). The shape of each $\alpha(y)$ expression is then a string in $(X_2 \cup X_s)^*$. We define the map g from $(Q_2 \times X_1 \times X_2)$ to $(X_2 \cup X_s)^*$, such that for each state $r \in Q_2$, and for each $x \in X_1$, $g(r, x, y)$ is the string that describes the shape of $\alpha(y)$. Thus, a state of T_s is the tuple $(q, f, g) \in (Q_1 \times [(Q_2 \times X_1) \to Q_2] \times [(Q_2 \times X_1 \times X_2) \to (X_2 \cup X_s)^*]$.

It can be shown that the maximum number of z-variables required for each state $r \in Q_2$ and each string variable $x \in X_1$ is $2.|X_2|$. Thus, the total number of string variables required for T_s is $2.|X_2|.|Q_2|.|X_1|$. Further, as the z and y variables strictly interleave and do not repeat, the number of possible shapes is finite. This is key to ensure that the size of T_s (number of states and string variables) is finite.

Example 3.1. Suppose that T_2 contains the transitions:

$t_1 : (r_0, a,\ (y_1, y_2) := (c.y_1.d,\ e.y_2),\ r_1)$ $\qquad$ $t_2 : (r_0, b,\ (y_1, y_2) := (y_1.y_2.g,\ \varepsilon),\ r_1)$
$t_3 : (r_1, b,\ (y_1, y_2) := (g.y_1.h,\ e.y_2.d),\ r_0)$

The state of T_s when T_1 is in state q and $\nu(x_1) = abb$ encodes the effect of T_2 executing transitions t_1, t_3 and t_2 (in that order), and is of the form (q, f, g). Here $f(r_0, x_1) = r_1$, $g(r_0, x_1, y_1) = z_1.y_1.z_2.y_2.z_3$, and $g(r_0, x_1, y_2) = z_4$. The valuation for the variables of T_s is: $\nu(z_1) = gc$, $\nu(z_2) = dhee$, $\nu(z_3) = dg$, and $\nu(z_4) = \varepsilon$. Similarly, if $\nu(x_2) = bbb$, we encode the effect of T_2 executing t_2, t_3, and t_2, and define $f(r_0, x_2) = r_0$, $g(r_0, x_2, y_1) = v_1.y_1.v_2.y_2.v_3$, and $g(r_0, x_2, y_2) = v_4$. The valuation of the variables is: $\nu(v_1) = g$, $\nu(v_2) = \varepsilon$, $\nu(v_3) = ghedg$, and $\nu(v_4) = \varepsilon$. □

Invariant. Suppose $\Delta_1(q_0^1, w)$ is (α, q) (*i.e.*, the summary of T_1 after reading w, beginning in its initial state q_0^1). Suppose the state of T_s after reading the input word w is (q_s, f, g) and the valuation of its string variables is ν_w. Then the following invariant holds: if $\Delta_1(q_0^1, w) = (\alpha, q)$, then $q_s = q$, and, if $\Delta_2(r, \nu_1(x)) = (\beta, r')$, then $f(r, x) = r'$, and for each y, $\nu_w(g(r, x, y)) = \beta(y)$.

Thus, to model the effect of a transition of T_1 of the form (q, a, α, q'), we add an edge of the form $\big((q, f, g),\ a, \alpha',\ (q', f', g')\big)$, where the maps f', g', and the assignment α' to the variables of T_s are defined in a manner that satisfies the above invariant.

Example 3.2. Consider the state in Ex. 3.1. Now suppose that T_1 executes the transition $(q, a,\ (x_1, x_2) := (x_2.x_1, \varepsilon),\ q')$. Suppose the resulting state of T_s is (q', f', g'). We only explain how the updates to the variable x_1 are encoded. To model the assignment $x_1 := x_2.x_1$, we define $f'(r_0, x_1) = f(f(r_0, x_2), x_1)$ which evaluates to r_1 (as seen from Ex. 3.1). Similarly, the map $g'(r_0, x_1, y_1)$ is obtained by sequentially composing the maps $g(r_0, x_2, y_1)$ and $g(r_0, x_1, y_1)$ from Ex. 3.1. The result of the composition yields the expression $\underline{z_1.v_1}.y_1.v_2.y_2.\underline{v_3.z_2.v_4.z_3}$, which is then compressed by combining adjacent z_i and v_i variables. Thus, $g'(r_0, x_1, y_1)$ is defined to be $z_1.y_1.v_2.y_2.z_3$ with the copyless assignments $z_1 := z_1.v_1$ and $z_3 := v_3.z_2.v_4.z_3$ (and no change to the other variables). Similarly, we can compute $g'(r_0, x_1, y_2) = z_4$ by looking at the maps $g(r_0, x_2, y_2)$ and $g(r_0, x_1, y_2)$. □

Sequential Composition of NSSTs. We now show that NSSTs are closed under sequential composition.

Theorem 3.1. *Let T_1 be a (Σ_1, Γ_1)-*NSST*, T_2 be a (Σ_2, Σ_3)-*NSST*, then one can effectively construct T_s, a (Σ_1, Σ_3)-*NSST *such that for all strings $w \in \Sigma_1^*$, we have $[\![T_s]\!](w) = \bigcup_{u \in [\![T_1]\!](w)} [\![T_2]\!](u)$.*

Proof. The construction for the sequential composition of DSSTs extends to the case of NSSTs in a straightforward fashion. We show how to compute the NSST $T_s = T_1 \circ T_2$. A run of T_s simulates a run of T_1 on the input string w, and simultaneously summarizes the action of some transition of T_2 on the contents of the string variables of T_1. This summary is stored with the help of the maps f and g that are part of its state, and its string variables. Thus, a state of T_s is a finite subset of $(Q_1 \times [f : Q_2 \times X_1 \to Q_2] \times [g : Q_2 \times X_1 \times X_2 \to (X_2 \cup X_s)^*])$.

Unlike the deterministic case where the computation summary is a pair, the summary $\Delta_2(r, \nu(x_1))$ of T_2 is the set of pairs $\{(\beta_1, r_1), (\beta_2, r_2), \ldots\}$. To accommodate this, we define an invariant that all states of T_s satisfy: If T_s is in the state (q, f, g) upon reading input string w, with the valuation ν_w for its string variables, then for some α, $(q, \alpha) \in \Delta_1(q_0^1, w)$, and, if $f(r, x) = r'$, and $\nu_w(g(r, x, y)) = \beta$, then $(\beta, r') \in \Delta_2(r, \nu_1(x))$. For a transition of the form (q, a, α, q') in T_1, we add a transition $((q, f, g), a, \alpha', (q', f', g'))$ to T_s, such that the copyless assignment α', and the maps f' and g' satisfy the invariant (for the input $w.a$). We finally remark that the number of variables of T_s is $2.|X_2|.|Q_1|.|X_1|$, (the same as for DSSTs) and as the maps f, g are finite, the number of states required for T_s is also finite. □

Theorem 3.2. *The transduction implemented by an* NSST *T is bounded-length,* i.e.*, there exists a constant c such that $\forall w \in \Sigma^*$, if $u \in [\![T]\!](w)$, then $|u| \leq c.|w|$.*

Proof. We use the notation $|\alpha(x)|$ to denote the number of output symbols in the expression $\alpha(x)$. We define $|\alpha| = \sum_x (|\alpha(x)|)$. Now let $\ell_{max} = \max_E(|\alpha|)$, *i.e.*, the maximum number of output symbols added by *any* transition in T. Let u_w be the shorthand for some output of T upon reading the string w. We prove the result by induction on the length of the input. The base case is valid; since if $|w| = 0$, the length of u_w is zero. The inductive hypothesis is stated as follows: for input w, if $u_w \in [\![T]\!](w)$, then $|u_w| \leq \ell_{max}.|w|$. Now consider an input of the form $w.a$, and let q be the state of T after reading w. Let E' be the subset of E containing transitions of the form (q, a, β, q'). The number of output symbols added by any transition in E' is bounded by $\max_{E'} |\beta|$. However, by definition, $\ell_{max} \geq \max_{E'} |\beta|$. Thus, for any $u_{w.a} \in [\![T]\!](w.a)$, $|u_{w.a}| \leq \ell_{max}.|w| + \ell_{max}$. Thus, $|u_{w.a}| \leq \ell_{max}.|w.a|$, since $|w.a| = |w| + 1$. □

3.2 Comparison with Existing Models

For classes of transductions C_1 and C_2, we denote by $C_1 \subseteq C_2$ the fact that for every transducer T of type C_1, there is a transducer T' of type C_2 such that

$[\![T]\!] = [\![T']\!]$. We say that C_1 is *equi-expressive* to C_2 if $C_1 \subseteq C_2$ and $C_2 \subseteq C_1$. The following relationships are known: DMSOS = 2DGSM [6], DMSOS = DSST [1], NMSOS $\not\subseteq$ 2NGSM and 2NGSM $\not\subseteq$ NMSOS [6]. We now compare the expressive power of NSSTs with respect to that of existing models/machines for nondeterministic transductions such as NMSOS-transductions, NGSM, and 2NGSM.

Expressive power of NGSM *vs.* NSST. A NGSM is a NSST with a single string variable (say x), and every transition is of the form $(q, a, x := x.\gamma, q')$, where $a \in \Sigma$, and $\gamma \in \Gamma^*$. Thus, NGSM $\subseteq$ NSST.

Expressive power of NMSOS *vs.* NSST. We now show that the the class of NSSTs is equi-expressive to NMSOS-transductions. We first show that for every NSST T we can define a NMSOS transduction W such that $[\![T]\!] = [\![W]\!]$. In [1], the authors shows how a sequence of states and copyless assignments of a DSST can be encoded in the copy set of a DMSOS-transduction. As a run of an NSST T is such a sequence of states and assignments, it can also be encoded in the copy set of a DMSOS-transduction. The key ideas in this construction are: (1) Let $n_x = \max_E |\alpha(x)|$ (*i.e.* the maximum number of output symbols added by any assignment to the string variable x). Then, assignments to x can be encoded within a copy set that has at most $n_x + 2$ copies. In any step, for each string variable x, we maintain a designated start vertex x^b and a designated end vertex x^e for x, and the valuation of x ($\nu(x)$) is obtained from the string graph corresponding to the path beginning at x^b and ending in x^e. This graph is linear and connected by construction. (2) Concatenation of string variables, say $x := x.y$, is encoded by adding an edge between x^e and y^b. (3) A state of the streaming transducer can be encoded by an MSO-formula on the input graph (these formulas essentially capture the next-state function). (4) The output is defined if the output graph – the sequence edges beginning in the start vertex for the output and concluding in the end vertex – is a string graph.

Lemma 3.1. NSST $\subseteq$ NMSOS.

Proof. Recall that a NMSOS transducer W has a finite copy set C, and a finite set of *parameters* $\mathcal{X} = \{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$, *i.e.*, second order variables ranging over sets of vertices. This proof combines two main ideas: (a) a given run of a NSST T can be encoded in the finite copy set of a NMSOS transducer W, and, (b) each run of T can be *guessed* by W by guessing a valuation for its parameters. The construction for part (a) is as discussed, for details see [1]. We focus on part (b).

As T is nondeterministic, there may be multiple transitions on a given input symbol that are enabled in a state q. However, as the assignment in each transition is a copyless assignment over the same set of string variables, the copy set required is the same as that for encoding a single run. However, different runs may label an edge in the copy set with different labels.

To make the NMSOS transduction W mimic a single run of T, we add parameters that consistently choose among the different labels available for a given set of edges. Let the maximum number of transitions of T from any state q on any input symbol a be d. Then, W uses the set of parameters $\mathcal{X} = \{\mathcal{X}_1, \ldots, \mathcal{X}_d\}$.

Intuitively, the j^{th} nondeterministic transition of T is chosen by W if the corresponding vertex of the input graph is present in the valuation chosen for $\mathcal{X}_j$ by W. Choosing the j^{th} transition fixes a particular assignment α_j to each string variable. Now for each edge formula $\varphi_a^{i,j}(x,y)$ that encodes the action of α_j, we define $\varphi_a^{i,j}(x,y,\mathcal{X}_1,\ldots,\mathcal{X}_d) = \varphi_a^{i,j}(x,y) \wedge (x \in \mathcal{X}_j)$. We can similarly encode the next state function in terms of the parameters.

We also assume that each of the vertex and edge formulas is conjoined with a domain formula that specifies that for any vertex $x \in G_i$, x is contained in at most one $\mathcal{X}_j$ in a given valuation, and the union of all $\mathcal{X}_j$ covers the set of all vertices of the input graph[3]. In other words, using the parameters in $\mathcal{X}$, W first *guesses* a possible sequence of choices at each state in T, and then encodes that sequence using the copy set and next-state relations (which are MSO-formulas as in the deterministic case). It is obvious that the number of parameters required is at most d, and thus finite. □

In Lemma 3.2, we show that for every NMSOS W, we can obtain a NSST T such that $[\![W]\!] = [\![T]\!]$. The key idea is that a NMSOS transduction is essentially the sequential composition of a string relabeling with a DMSOS transduction, both of which are expressible as NSSTs.

Lemma 3.2. NMSOS $\subseteq$ NSST.

Proof. We recall from [6], that any NMSOS transduction W can be written as the sequential composition of a string relabeling and a DMSOS transduction. A string relabeling nondeterministically substitutes each symbol in w with a finite string in Γ^*. A string relabeling can be easily encoded by an NSST T_1 that has one state q, one string variable x, and for all $a \in \Sigma$, its transitions are of the form $(q, a, x := x.\gamma, q)$, where γ is a finite string in Γ^*. From [1], we know that for every DMSOS W, there exists a DSST T_2 that implements the same transduction. Every DSST is an NSST, and from Theorem 3.1 we know that NSSTs are closed under sequential composition; thus, $T = T_1 \circ T_2$ is a NSST s.t. $[\![W]\!] = [\![T]\!]$. □

From Lemmas 3.1 and 3.2, we can conclude that NSSTs are equi-expressive to NMSOS-transductions, formalized in Theorem 3.3 below.

Theorem 3.3. NSST = NMSOS.

Comparing ε-NSSTs with existing models. We now investigate the expressive power of ε-NSSTs, *i.e.*, NSSTs that are allowed transitions without consuming any input symbol. The expressive power of ε-NSSTs is greater than that of NSSTs: consider an ε-NSST in which there is a state q reachable from the initial state q_0 on the input string w, and q has an ε-loop (a cycle with only ε-transitions) that adds some symbols to the output. The set of outputs for any input with w as its prefix is infinite, as the ε-NSST can cycle through the ε-loop an infinite number of times. Thus, the transduction realized by a general ε-NSST is nonfinitary.

[3] It further ensures the technicality that if the number of outgoing transitions of T corresponding to a particular state and input symbol is d', where $d' < d$, then in this case, all $\mathcal{X}_j$ where $j > d'$ always evaluate to the empty set.

If we restrict ε-NSSTs so that they are ε-loop-free, then we can show that the class of such transducers coincides with NSSTs. The proof uses a construction similar to the one used for eliminating ε-transitions in NFAs: a finite sequence of transitions of the form $(q_1, \varepsilon, \alpha_1, q_2)$, ..., $(q_n, \varepsilon, \alpha_n, q_{n+1})$, (q_{n+1}, a, β, q') can be replaced by the single transition: $(q_1, a, \alpha_1 \circ \ldots \circ \alpha_n \circ \beta, q')$. Since the transducer is ε-loop-free, an infinite sequence with ε-transitions does not exist.

We now compare ε-NSSTs with 2NGSMs. Though both classes can implement nonfinitary transductions, we show that their expressive power is incomparable. We first note that NMSOS $\not\subseteq$ 2NGSMs [6], and as NSST = NMSOS (Theorem 3.3), and NSST $\subseteq$ ε-NSST, we have that ε-NSST $\not\subseteq$ 2NGSM, *i.e.*, there are transductions implemented by NSSTs and ε-NSSTs that cannot be implemented by 2NGSMs. We further show that 2NGSM $\not\subseteq$ ε-NSST, by an example transduction that can be realized by a 2NGSM but cannot be realized by an ε-NSST. A similar example is used in [6] to show the incomparability of NMSOS with 2NGSM; however, the argument used therein is not applicable in our setting, as the transductions realized by both 2NGSMs and ε-NSSTs are nonfinitary.

Theorem 3.4. *The transduction* $R_{mult} = \{(w, w^m) \mid m \geq 1, w \in \Sigma^*\}$ *can be realized by a* 2NGSM*, but cannot be realized by an* ε-NSST*.*

Proof. The 2NGSM implementing the transduction R_{mult} has three states q, q_b and *halt*. It starts in state q scanning the input $\vdash w \dashv$ from left to right, simultaneously copying it to the output. Upon reaching $\dashv$, it either transitions to the state *halt* and moves right from $\dashv$, or transitions to q_b and scans the input from right to left without producing any output. Once it reaches $\vdash$, it transitions back to the state q.

We note that R_{mult} is a nonfinitary transduction. If there is a ε-NSST T_ε that implements it, it must have a ε-loop on some state q that is reachable from the initial state q_0. We denote the summary of T_ε for the ε-loop by $\Delta(q, \varepsilon\text{-loop})$. Suppose $\Delta(q, \varepsilon\text{-loop}) = (\beta, q)$. Let $u, v \in \Sigma^*$ be substrings of w such that $u.v = w$, u is the input that drives T_ε from q_0 to q, and v is the input that drives T_ε from q to some state q' (possibly the same), where $F(q')$ is defined. Now let $\Delta(q_0, u) = (\alpha_1, q)$, $\Delta(q, v) = (\alpha_2, q')$. Then $\Delta(q, w)$ is the set $\{(\alpha_1 \circ \alpha_2, q'), (\alpha_1 \circ \beta \circ \alpha_2, q'), (\alpha_1 \circ \beta \circ \beta \circ \alpha_2, q'), \ldots\}$.

Suppose the number of symbols added to the output by α_1 is t_1, by β is t, and by α_2 is t_2. Then the lengths of the outputs of T_ε in state q' are $t_1 + t_2$, $t_1 + t + t_2$, $t_1 + 2.t + t_2$, and so on. However, R_{mult} requires that for *any* input w, each of the outputs have length that is a multiple of $|w|$. This is possible only if $t_1 = t_2 = 0$ and $t = |w|$. However, t is a fixed number for a given ε-loop, and independent of the input w. Thus, once we fix T_ε, it will produce incorrect outputs for any w where $|w| \neq t$. □

Functional NSSTs. It is clear that the expressive power of functional NSSTs is identical to that of the class DSST, as a transduction realized by a functional NSST is single-valued and MSO-definable, *i.e.*, a DMSOS transduction, which is the same as that for a DSST.

Theorem 3.5. *Functional* NSST = DSST.

It is also easy to show that functional NSSTs are closed under sequential composition. The results follow from the fact that NSSTs are closed under sequential composition, and single-valued transductions are closed under sequential composition. Finally, the expressivity relations between the different transducer models are as shown in Fig. 1.

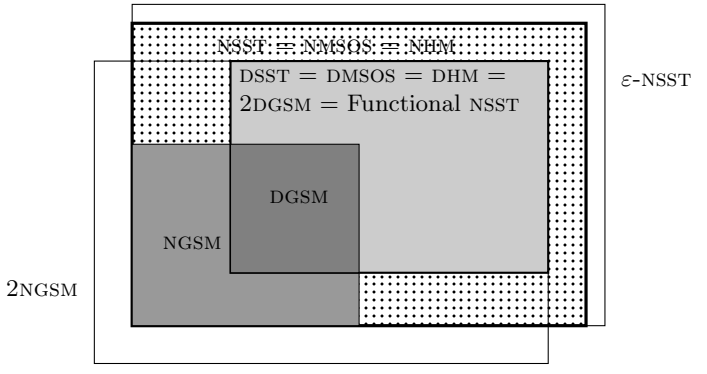


Fig. 1. Expressivity Results

4 Decision Problems

In this section, we discuss decision problems for NSSTs, with special heed to the class of *functional* NSSTs. In what follows, we make use of 1-*reversal-bounded* m-*counter machines*. Such a machine is an NFA augmented with m counters, where each counter can make at most one reversal (*i.e.*, in any computation each counter can go from increasing to decreasing mode at most once). It has been shown that the nonemptiness problem for such machines is in NLOGSPACE in the size of the machine's description. For further details on reversal-bounded counter machines, please see [8].

Checking functionality. We now address the problem of checking if an arbitrary NSST is functional. We show that this problem is decidable, and present an algorithm with PSPACE complexity. To a large extent, our algorithm draws upon techniques developed for checking equivalence of DSSTs discussed in [2]. We use the following intuition: An NSST is not functional if there exists some input string w such that there are outputs $y_1, y_2 \in [\![T]\!](w)$ and $y_1 \neq y_2$. We reduce the problem of finding such distinct outputs to the nonemptiness problem for a 1-reversal-bounded 2-counter machine.

Theorem 4.1. *For an* NSST T*, checking if* T *is functional is in* PSPACE.

Proof. We construct a 1-reversal-bounded 2-counter machine M that for a given input string w simultaneously simulates two paths in T corresponding to w,

and checks if the resulting outputs u_1 and u_2 differ because of either of these conditions: (1) $|u_1| \neq |u_2|$, *i.e.*, the lengths of u_1 and u_2 are different, (2) there exists a position p such that the p^{th} symbol in the string u_1 (denoted $u_1[p]$) differs from $u_2[p]$. M nondeterministically chooses one of the above conditions to check. Checking the first condition is easier and uses similar techniques as for checking the second condition; thus, we focus on the latter. We now explain how the second condition is checked.

M precisely keeps track of the state of T in each of the two paths being simulated in its state. At the beginning of the computation, M pre-loads its counters c_1 and c_2 with the same number; this number is the guess for the position p in which the outputs might differ. For path i ($i = 1, 2$), at some step in the simulation, M nondeterministically guesses that the symbol γ_i added by a copyless assignment in this step appears at the p^{th} position in the corresponding output u_i, and remembers γ_i in its state. It also guesses where each string variable appears with respect to p in this output. Concretely, we define the mapping ϱ (called the string variable class) from X to an element of $\zeta = \{L, R, C, N\}$. Here, $\varrho(x)$ is L, R, C or N, depending on whether x appears to the left of p, to the right of p, contains p, or does not contribute to the output respectively. For each output symbol added to the left of p in the first path, M decrements the counter c_1, and for each output symbol added to the left of p in the second path, M decrements c_2.

Let the symbol $\perp$ denote that M has not made its guess for the symbol in position p. Let $\Gamma' = \Gamma \cup \{\perp\}$. Formally, the set of states S of M is a subset of $(Q \times Q \times (\Gamma' \times \zeta^X) \times (\Gamma' \times \zeta^X))$. To elaborate, the meaning of some state $(q, q', \gamma_1, \varrho_1, \perp, \varrho_2)$ in M is: T reaches states q and q' from its initial state on the same input string; M has guessed that the symbol at $u_1[p]$ is γ_1, M has not yet guessed the symbol at $u_2[p]$, and the mappings from the string variables to classes along the two paths are ϱ_1 and ϱ_2. Except for the initial transitions where M pre-loads c_1 and c_2, all transitions of M decrement the counters by some non-negative amount. Thus, the counters of M are single-reversal.

The transitions of M are defined in a way that ensures that M consistently guesses the classes for each string variable. We illustrate this with two examples. Suppose $(q, a, (x, y) := (x.d.y,\ b), r)$ and (q', a, α, r') are two transitions in T.

Suppose in state r, M guesses that the symbol d added to the string variable x appears at $u_1[p]$. In order to maintain consistency, in state q, M must have assigned x to the class L, and y to the class R. In other words, we add a transition from the state $(q, q', \perp, [\varrho_1(x) = L, \varrho_1(y) = R], \gamma_2, \varrho_2)$ to the state $(r, r', d, [\varrho'_1(x) = C, \varrho'_1(y)], \gamma_2, \varrho'_2)$, with no change to the counters. The mapping $\varrho'_1(y)$ maps y to any string class in ζ other than C, and the mappings ϱ_2 and ϱ'_2 similarly ensure consistency with the assignment α along the second path.

Now consider the case where in state r, M guesses that the contents of x (which are now $x.d.y$) appear to the left of the position p in u_1. In order to maintain consistency, in state q, M must have assigned x and y to the class L. Thus, we add a transition between the states $(q, q', \gamma_1, [\varrho_1(x) = L, \varrho_1(y) = L], \gamma_2, \varrho_2)$ and $(r, r', \gamma_1, [\varrho'_1(x) = L, \varrho'_1(y)], \gamma_2, \varrho'_2)$ in M. As the assignment adds

one letter to the left of p, M decrements c_1 by one. Here, $\varrho'_1(y)$ maps y to any string class in ζ, and γ_1 is propagated unchanged.

Initially, no string variable is guessed to be in the class C, and at each step at most one string variable can be guessed to be in the class C. At the end of a computation, if $c_1 = c_2 = 0$, then it means that the symbols γ_1 and γ_2 in the state of M are symbols in the same position p in u_1 and u_2, *i.e.*, $\gamma_1 = u_1[p]$ and $\gamma_2 = u_2[p]$. We define a state $(q, q', \gamma_1, \varrho_1, \gamma_2, \varrho_2)$ of M as *accepting* if (1) $c_1 = c_2 = 0$, (2) $\gamma_1 \neq \gamma_2$, (3) if for some $x \in X$, if $\varrho_1(x) = C$ then $F(q) = x$[4], and (4) if for some $y \in X$, if $\varrho_2(y) = C$ then $F(q') = y$. M thus accepts only those paths that are witnesses to an input string that leads to two distinct output strings. In other words, T is functional iff M is empty. The problem of checking nonemptiness of 1-reversal bounded m-counter machines is in NLOGSPACE in the size of M. As the total number of states of M is $(|Q|^2.(|\Gamma| + 1)^2.|\zeta|^{2.|X|})$, *i.e.*, exponential in the size of X, the above technique gives us a procedure that is in PSPACE. □

Equivalence Checking. We now discuss the equivalence problem for NSSTs. Given two (Σ, Γ)-NSSTs T_1 and T_2, the equivalence problem is to determine if for all inputs $w \in \Sigma^*$, $[\![T_1]\!](w) = [\![T_2]\!](w)$. We start with a negative result that shows that for the general case of NSSTs, checking equivalence is undecidable.

Theorem 4.2. *The equivalence problem for* NSSTs *is undecidable.*

Proof. As mentioned in Section 3.2, an NGSM is a special case of a NSST. The result follows from the fact that the equivalence problem for NGSMs is known to be undecidable by a reduction from the Post Correspondence problem [7]. □

If we restrict our attention to functional NSSTs, the equivalence problem is decidable and a minor modification of the algorithm for checking equivalence of DSSTs in [2] can be directly applied to give a PSPACE complexity procedure. Moreover, we show that the problem is in fact PSPACE-COMPLETE.

Theorem 4.3. *The equivalence problem for the class of functional* NSSTs *is* PSPACE-COMPLETE.

Proof. Membership in PSPACE can be shown by using an algorithm similar to the one for checking equivalence of DSSTs [2].

We further show that the problem is PSPACE-HARD by a reduction from the equivalence problem for NFAs. We can encode an NFA as an NSST T which uses only one string variable x and never changes the value of x from the initial value of ε. The states of the NFA are the same as the states of T. We define the output function F of T such that for each final state q_f of the NFA, in the corresponding

[4] Here, we assume that, for all q, $F(q)$ is defined to be some string variable x. An NSST in which $F(q)$ is an arbitrary expression $\alpha \in (X \cup \Gamma)^*$ can be converted to an equivalent NSST of this form as follows: add a new state $q_f \notin Q$; for all q where $F(q) = \alpha$, add a transition from q to q_f with the copyless assignment $x := \alpha$ on that transition; finally, define $F(q_f) = x$, and $F(q)$ as undefined.

state of T, $F(q_f) = x$, and for all other states q of the NFA, $F(q)$ is undefined. Clearly, $[\![T]\!](w) = \{\varepsilon\}$ implies that w is accepted by the NFA. Observe that the NSST defined thus is functional: for all inputs w, $[\![T]\!](w)$ is either empty or contains exactly one symbol. If two such NSSTs T_1 and T_2 are equivalent, it means that either (1) there exist runs of T_1 and T_2 in which both read the same input string w, and produce the same output ε, or (2) all runs of T_1 and T_2 reading the same input string w reach some states q and q' respectively, such that $F_1(q)$ and $F_2(q')$ are undefined. In other words, two such NSSTs are equivalent iff the encoded NFAs are equivalent. □

Containment. The problem of checking if the transduction realized by an NSST is contained within the outputs of a finite set of DSSTs is of particular interest in applications such as the verification of concurrent heap-manipulating methods. Given an NSST T, and DSSTs $D_1,\ldots,D_n$, we show that the problem of checking if the set of outputs of T is contained within the set of outputs of $D_1, \ldots, D_n$ is in PSPACE by reducing the problem to the nonemptiness problem for 1-reversal m-counter machines.

Theorem 4.4. *Suppose $D_1, \ldots, D_n$ are (Σ, Γ)-DSSTs, and T is a (Σ, Γ)-NSST. Checking if $[\![T]\!] \subseteq \bigcup_{j=1}^{n} [\![D_j]\!](w)$ is in* PSPACE.

Proof. We observe that $[\![T]\!] \not\subseteq \bigcup_{j=1}^{n} [\![D_j]\!](w)$, if there is a run of T on the input string w that produces an output u_t such that for all the DSSTs D_j, either the output of D_j on w is undefined, or the output of D_j is different from u_t. Formally,

$$\exists w \in \Sigma^*, \exists u_t \in [\![T]\!](w) : \bigwedge_{j=1}^{n} \begin{pmatrix} [\![D_j]\!](w) \text{ is not defined} \vee \\ |u_t| \neq |[\![D_j]\!](w)| \quad \vee \\ \exists p_j : u_t[p_j] \neq [\![D_j]\!](w)[p_j] \end{pmatrix} \tag{4.1}$$

We can expand the above condition to obtain a disjunction over different combinations of conjunctions of the inner disjuncts. Each term in this disjunct illustrates a way in which u_t is not generated by all of the n DSSTs. We construct a 1-reversal $(2.n)$-counter machine M that nondeterministically decides to check one of the conjunctive terms in (4.1). In particular, we show how M checks the conjunction: $\bigwedge_{j=1}^{n} (u_t[p_j] \neq [\![D_j]\!](w)[p_j]))$, the other conjunctions are simpler and can be checked by similar techniques.

M simulates the action of $D_1, \ldots, D_n$, and T in parallel. It precisely keeps track of the states of each of these transducers in its state. For each pair (D_j, T), M maintains a pair of counters (c_{jt}, c_{tj}). For each of these n pairs, M pre-loads the counters with some number p_j (the same for both), which is the guess of M for the position p_j such that $u_t[p_j] \neq [\![D_j]\!](w)[p_j]$. At some point in its simulation of T (resp. D_j), M guesses that it has output the symbol $\gamma_{tj} = u_t[p_j]$ (resp. $\gamma_{jt} = [\![D_j]\!](w)[p_j]$) and remembers it in its state. It also guesses where each string variable of T (resp. D_j) appears with respect to the position p_j, thus mapping each string variable to a class in $\zeta = \{L, C, R, N\}$. For each output symbol added by T (resp. D_j) to the left of p_j, M decrements the counter c_{tj} (resp. c_{jt}) by one.

Let $\Gamma' = \Gamma \cup \{\perp\}$, where $\perp$ is the symbol indicating that M has not guessed the symbol at position p_j yet. The set of states of M is a subset of $((Q_T \times Q_1 \times \ldots \times Q_n) \times (\Gamma'^2 \times \zeta^{X_T} \times \zeta^{X_1}) \times \ldots \times (\Gamma'^2 \times \zeta^{X_T} \times \zeta^{X_n}))$. The transitions of M preserve consistency in its guesses for the string variable classes, using the same technique as in the proof of Theorem 4.1. Note that apart from the initial transitions of M that pre-load the counters, all transitions decrement the counters by some non-negative number; thus M has only one reversal.

Once M reaches the end of the input string w, it checks if the symbol guessed for D_j and T is at the same position p_j, by checking if $c_{tj} = c_{jt} = 0$. We define a state as *accepting* if *for all* j, $c_{jt} = c_{tj} = 0$, and for each pair of symbols recorded in the state, $\gamma_{jt} \neq \gamma_{tj}$. In other words, M accepts a computation iff it encodes the parallel action of T, D_1,...,D_m on an input string w, producing an output u_t in $[\![T]\!](w)$ such that u_t is different from each of the n outputs of the transducers $D_1, \ldots, D_n$. Thus $[\![T]\!] \subseteq \bigcup_{i=1}^{n} [\![D_i]\!]$ iff M is empty. From [8], we know that checking the nonemptiness of a 1-reversal m-counter machine is in NLOGSPACE (in the size of M). For a fixed n, the number of states of M is dominated by an exponential in the size of the largest set among $X_T, X_1, \ldots, X_n$. Thus, the above construction yields us a procedure that is in PSPACE. □

5 Discussion

Checking k-valuedness. A k-valued NSST naturally extends a functional NSST. This class is not closed under sequential composition because given two k-valued transducers T_1 and T_2, the NSST $T_1 \circ T_2$ could be k^2-valued. Checking if an arbitrary NSST T is k-valued is decidable. We skip the proof in interest of brevity. Similar to the proof of Theorem 4.1, the basic idea is to reduce k-valuedness of an NSST to the nonemptiness problem for a 1-reversal-bounded $(k.(k+1))$-counter machine M that detects if there is some input w on which T produces $(k+1)$ distinct outputs. M simulates $(k+1)$ copies of T in parallel, and for each pair of the possible $(k+1)$ outputs, M uses a pair of counters to check if the outputs are different. M is empty iff T is k-valued. Our approach can be viewed as an extension of the algorithm to check the k-valuedness of NFTs [9].

Equivalence of k-valued NSSTs. The equivalence problem for k-valued NFTs has been shown to be decidable in [5], and more recently in [13]. We are interested in investigating the same problem for k-valued NSSTs. The approach in [5] reduces the equivalence of k-valued NFTs to the solution of an infinite system of word equations, which by the validity of Ehrenfeucht's conjecture, is equivalent to a finite subsystem. The authors then show how the finite subsystem can be effectively characterized for NFTs, which leads to a decision procedure. The proof of [5] relies on the fact that the valuedness of an NFT is at the most its maximum edge-ambiguity[5]. However, as the maximum edge-ambiguity of an NSST does not

[5] The edge-ambiguity of a transducer is the maximum number of transitions between any two states q and q' that have the same input symbol.

place such a bound on its valuedness, this proof strategy fails for proving the decidability of equivalence checking for k-valued NSSTs. In [13], the authors rely on a procedure to decompose a k-valued NFT T into k functional NFTs whose union is equivalent to T. Whether such a decomposition can be generalized to NSSTs remains open.

Bounded-valued NSST*s.* The class of bounded-valued NFTs has been extensively studied in the literature [14], [11]. The class of bounded-valued NSSTs promises to be a robust class. Unlike the class of k-valued NSSTs, it is closed under sequential composition. In both [14] and [11], it is shown that the problem of checking if an NFT is bounded-valued can be checked in polynomial time by verifying certain conditions on the structure of its transition diagram. We believe that it may be possible to generalize these conditions to characterize bounded-valuedness of NSSTs. However, the extension is not straightforward due to the presence of multiple string variables, and allowing output symbols to be appended to the left as well as to the right.

In summary, some of the interesting open problems are:

- Is the equivalence problem for k-valued NSSTs decidable?
- Given a k-valued NSST T can it be effectively decomposed into k functional NSSTs $T_1,\ldots,T_k$, such that $[\![T]\!] = \bigcup_{i=1}^{k} [\![T_i]\!]$?
- Is the bounded-valuedness of NSSTs decidable?

Applications. One of the motivations for studying NSSTs is their potential as models for verifying concurrent heap-manipulating programs. In [3], the authors study the problem of verifying linearizability of methods that modify linked-lists. Checking linearizability of a pair of methods m_1 and m_2 involves verifying if the interleaved execution of m_1 and m_2 (denoted by $m_1 \parallel m_2$) finishes with the same linked-list as executing m_1 followed by m_2 (denoted by $m_1; m_2$) or m_2 followed by m_1 ($m_2; m_1$). The authors show that with a specific regimen for pointer updates this problem can be algorithmically solved.

In [2], the authors have demonstrated how *sequential* heap-manipulating methods can be modeled as DSSTs. Proving linearizability of methods m_1 and m_2 is thus the same as verifying if the interleaved product of the corresponding DSSTs D_1 and D_2 is contained within their union, *i.e.*, checking if $[\![D_1 \parallel D_2]\!] \subseteq ([\![D_1 \circ D_2]\!] \vee [\![D_2 \circ D_1]\!])$. In general, it may not be possible to model the interleaved product of DSSTs as an NSST. However, if we identify a subclass of DSSTs such their interleaved product is an NSST, then checking linearizability reduces to checking containment of this NSST in the finite union of DSSTs (each of which represents a possible sequential execution of the given DSSTs). We have shown that checking containment is decidable in PSPACE. Thus, we believe that NSSTs could serve as a foundational model of *concurrent* heap-manipulating methods, by helping us better understand the theoretical limits of decidability of the verification problems for such methods.

References

1. Alur, R., Černý, P.: Expressiveness of streaming string transducers. In: Proc. of Foundations of Software Technology and Theoretical Computer Science, pp. 1–12 (2010)
2. Alur, R., Černý, P.: Streaming transducers for algorithmic verification of single-pass list-processing programs. In: Proc. of Principles of Programming Languages, pp. 599–610 (2011)
3. Černý, P., Radhakrishna, A., Zufferey, D., Chaudhuri, S., Alur, R.: Model checking of linearizability of concurrent list implementations. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 465–479. Springer, Heidelberg (2010)
4. Courcelle, B.: Graph Operations, Graph Transformations and Monadic Second-Order Logic: A survey. Electronic Notes in Theoretical Computer Science 51, 122–126 (2002)
5. Culik, K., Karhumäki, J.: The equivalence of finite valued transducers (on HDT0L languages) is decidable. Theor. Comp. Sci. 47, 71–84 (1986)
6. Engelfriet, J., Hoogeboom, H.J.: MSO definable String Transductions and Two-way Finite-State Transducers. ACM Transactions on Computational Logic 2(2), 216–254 (2001)
7. Griffiths, T.V.: The unsolvability of the equivalence problem for ε-Free nondeterministic generalized machines. Journal of the ACM 15, 409–413 (1968)
8. Gurari, E.M., Ibarra, O.H.: The Complexity of Decision Problems for Finite-Turn Multicounter Machines. J. Comput. Syst. Sci. 22(2), 220–229 (1981)
9. Gurari, E.M., Ibarra, O.H.: A Note on Finite-valued and Finitely Ambiguous Transducers. Mathematical Systems Theory 16(1), 61–66 (1983)
10. Gusfield, D.: Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, Cambridge (1997)
11. Sakarovitch, J., de Souza, R.: On the decidability of bounded valuedness for transducers. In: Proc. of Mathematical Foundations of Computer Science, pp. 588–600 (2008)
12. Schützenberger, M.P.: Sur les relations rationelles entre monoïdes libres. Theor. Comput. Sci., 243–259 (1976)
13. de Souza, R.: On the Decidability of the Equivalence for k-Valued Transducers. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 252–263. Springer, Heidelberg (2008)
14. Weber, A.: On the Valuedness of Finite Transducers. Acta Informatica 27(8), 749–780 (1990)

An Introduction to Randomness Extractors

Ronen Shaltiel*

University of Haifa

Abstract. We give an introduction to the area of "randomness extraction" and survey the main concepts of this area: deterministic extractors, seeded extractors and multiple sources extractors. For each one we briefly discuss background, definitions, explicit constructions and applications.

1 Introduction

Randomized algorithms and protocols play an important role in many areas of computer science. It is often the case that such algorithms and protocols are more efficient than deterministic ones. Moreover, having access to randomness is essential for Cryptography.

Randomized algorithms and protocols are designed under the assumption that computers have access to a sequence of truly random bits (that is a sequence of independent and unbiased coin tosses). In actual implementations this sequence is generated by taking a sample from some "source of randomness". Examples are:

- Generating and measuring electromagnetic or radioactive noise.
- Measuring timing of past events (e.g., how much time did the last disk operation take?).
- Measuring user dependent behavior (e.g., timing of the key strokes of users).

While these sources seem to "contain randomness" in the sense that they have entropy, a sample from such sources is not of the form of truly random bits. Randomness extractors are algorithms that when given one sample from a weak random source, produce a sequence of truly random bits.

The motivation described above led to a wide body of research concentrating on three main concepts:

- Deterministic extractors, which we discuss in Section 2.
- Seeded extractors, which we discuss in Section 3.
- Multiple source extractors, which we discuss in Section 4.

It turns out that extractors have many applications beyond the original motivation presented in Section 1. We present few of these applications as we go along (and stress that there are many other applications in the literature).

Our aim is to provide a brief introduction to the area. This article does not attempt to be comprehensive and the reader is referred to [NTS99, Sha02, Vad07, AB09] for some other survey articles.

* The author is supported by ISF grant 686/07.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 21–41, 2011.

2 Deterministic Extractors

In this section we discuss "deterministic extractors". The term "deterministic" is used to distinguish these extractors from "seeded extractors" that we discuss in Section 3. We begin with some notation. Throughout this manuscript we use the terms "source" and "distribution" interchangeably.

Definition 1 (Statistical distance). *Two distributions X, Y over the same domain are* ϵ-close *if for every event A, $|\Pr[X \in A] - \Pr[Y \in A]| \leq \epsilon$. The support of a distribution X is $\mathsf{Supp}(X) = \{x : \Pr[X = x] > 0\}$. The uniform distribution over $\{0,1\}^m$ is denoted by U_m and we say that X is ϵ-close to uniform if it is ϵ-close to U_m.*

Two distributions that are ϵ-close assign essentially the same probability to all events. In particular, randomized algorithms and protocols retain their useful properties when run with distributions that are close to uniform (rather than uniform). The motivation given in Section 1 leads to the following formal definition of an extractor (we also define a weaker object called a "disperser").

Definition 2 (deterministic extractors and dispersers). *Let $m \leq n$ be integers and let $\epsilon \geq 0$ be a parameter. Let $E : \{0,1\}^n \to \{0,1\}^m$ be a function and X be a distribution over $\{0,1\}^n$.*

- *E is an* ϵ-extractor *for X if the distribution $E(X)$ is ϵ-close to U_m.*
- *E is an* ϵ-disperser *for X if $|\mathsf{Supp}(E(X))| \geq (1-\epsilon) \cdot 2^m$.*

Let $\mathcal{C}$ be a class of probability distributions over $\{0,1\}^n$.

- *E is an* ϵ-extractor *for $\mathcal{C}$ if E is an ϵ-extractor for every X in $\mathcal{C}$.*
- *E is an* ϵ-disperser *for $\mathcal{C}$ if E is an ϵ-disperser for every X in $\mathcal{C}$.*

Note that every ϵ-extractor is in particular an ϵ-disperser. We plan to extract randomness from weak random sources and use this randomness in randomized algorithms and protocols. In the scenario described in Section 1 the "computer designer" can choose an implementation of the weak random source. Nevertheless, note that in the examples given there, this does not necessarily determines the distribution of the source (as the environment in which the computer operates may change). This leads to the following goal.

Goal: Design extractors for "large" families of "interesting" sources.

2.1 Min-entropy: Measuring the Number of Random Bits in a Source

Let us start with a simple observation. If $E : \{0,1\}^n \to \{0,1\}^m$ is a 0-extractor for X then for every $x \in \mathsf{Supp}(X)$, $\Pr[X = x] \leq 2^{-m}$. (As otherwise, for an x' with $\Pr[X = x'] > 2^{-m}$, we have that $\Pr[E(X) = E(x')] > 2^{-m}$ contradicting the correctness of the extractor as $\Pr[U_m = E(x')] = 2^{-m}$ and the two distributions $E(X)$ and U_m assign different probabilities to some event). Thus, a necessary condition for extracting m random bits from a distribution X is that for every $x \in \mathsf{Supp}(X)$, $\Pr[X = x] \leq 2^{-m}$. This leads to the following concept of entropy.

Definition 3 (min-entropy). *Let X be a distribution. The* min-entropy *of X (denoted by $H_\infty(X)$) is $H_\infty(X) = min_{x \in \mathsf{Supp}(X)} \log \frac{1}{\Pr[X=x]}$.*

We use min-entropy to measure the amount of random bits that can be extracted from a source.[1] Note that a distribution with min-entropy at least m has that for every $x \in \mathsf{Supp}(X)$, $\Pr[X = x] \leq 2^{-m}$. By the previous discussion having min-entropy at least m is a necessary condition for extracting m bits of randomness.[2] We could hope that it is a sufficient condition and that there exists an extractor $E : \{0,1\}^n \to \{0,1\}^m$ for all distributions with min-entropy at least m. However, this does not hold. In fact, for every function $E : \{0,1\}^n \to \{0,1\}$ there exists a distribution X over $\{0,1\}^n$ such that $H_\infty(X) \geq n-1$ and yet $E(X)$ is completely fixed. (For this, take X to be the uniform distribution over $S = \{x : E(x) = b\}$ for $b \in \{0,1\}$ which gives $|S| \geq 2^n/2$).

Summing up, we cannot have an extractor that extracts even a single bit from all distributions with very large min-entropy. Furthermore, if we plan to use function E as an extractor for $\mathcal{C}$, we cannot allow distributions that are uniform on $\{x : E(x) = b\}$ to be in the family $\mathcal{C}$.

2.2 Explicitness

By the previous discussion, deterministic extractors and dispersers $E : \{0,1\}^n \to \{0,1\}^m$ only exist for classes $\mathcal{C}$ of sources with some "special structure" where each X in $\mathcal{C}$ has $H_\infty(X) \geq m$. By the probabilistic method it is easy to show existence of extractors for such classes $\mathcal{C}$ which contain "few sources".

Existence of deterministic extractors: Let $m \leq n$ be integers, let $\epsilon > 0$ and let $\mathcal{C}$ be a class with at most $2^{\mathrm{poly}(n/\epsilon)}$ distributions over $\{0,1\}^n$. There exist $k = m + O(\log n + \log(1/\epsilon))$ such that if every X in $\mathcal{C}$ has $H_\infty(X) \geq k$ then there exists $E : \{0,1\}^n \to \{0,1\}^m$ that is an ϵ-extractor for $\mathcal{C}$.

However, for our intended application (as well as other applications that we will consider) we require extractors that can be efficiently computed. In this article we identify efficient computation with polynomial-time and this leads to the following definition of explicitness.

[1] It is natural to compare min-entropy with the more standard Shannon entropy given by $H(X) = \sum_{x \in \mathsf{Supp}(X)} \Pr[X = x] \cdot \log \frac{1}{\Pr[X=x]}$. Note that $H_\infty(X) \leq H(X)$ and equality holds for "flat distributions" (that are uniform over their support). Loosely speaking, min-entropy measures the amount of information in a distribution on the "worst case" when taking a single sample, while Shannon entropy measures the amount of information that a distribution has "on average" when taking many independent samples. Following this intuition, min-entropy is the right choice for our setup as we are interested in the behavior of extractors on a single sample from the source distribution.

[2] The previous discussion only considered 0-extractors. However, it is easy to check that if $E : \{0,1\}^n \to \{0,1\}^m$ is an ϵ-extractor for X then X is ϵ-close to some distribution Y with $H_\infty(Y) \geq m$. Thus, a necessary condition for extracting m random bits from a distribution X is that X is ϵ-close to some distribution with min-entropy at least m.

Definition 4 (Explicit extractors and dispersers). *Let $m(\cdot), \epsilon(\cdot)$ be functions over integers. A function E from strings to strings is an* explicit $\epsilon(\cdot)$*-extractor (resp. disperser) if it can be computed in polynomial time, and for every sufficiently large n, when restricted to inputs of length n, E outputs strings of length $m(n)$ and is an $\epsilon(n)$-extractor (resp. disperser).*

In the remainder of this section we survey some of the families of sources that are considered in the literature on deterministic extractors and dispersers.

2.3 Deterministic Extractors for von-Neumann Sources

The notion of deterministic extractors can be traced back to von-Neumann [vN51] who considered sequences of independent tosses of a biassed coin with unknown bias.

Definition 5. *A distribution X over $\{0,1\}^n$ is a vN-source with probability threshold $p_0 > 0$ if $X_1, \ldots, X_n$ are independent, and there exists $p_0 \leq p \leq 1 - p_0$ such that for every $1 \leq i \leq n$, $\Pr[X_i = 1] = p$.*

In order to extract one bit from such sources, von-Neumann [vN51] divided the n input bits into pairs. The extractor scans the pairs one by one and if it finds a pair of bits that are different, it stops and outputs the first bit in the pair.[3] The correctness of this scheme follows from the observation that for two independent coin tosses X_1, X_2 of a coin with bias p, $\Pr[X_1 = 0 \wedge X_2 = 1] = \Pr[X_1 = 1 \wedge X_2 = 0] = p \cdot (1 - p)$. Moreover, the probability that $n/2$ independent pairs do not produce an output bit is bounded by $(p_0^2 + (1 - p_0)^2)^{n/2} \leq \epsilon$ if $p_0 \geq \frac{\log(1/\epsilon)}{n}$. It is easy to extend this approach to extract many bits. There is also work on extracting a number of bits that approaches the information theoretic limit [Eli72, Per92].

2.4 Impossibility of Extraction from Santha-Vazirani Sources

It is natural to try and relax the assumption of independence between bits in a vN-source. Santha and Vazirani [SV86] considered a generalization of a vN-source in which for every $1 \leq i \leq n$, and every "prefix" $x_1, \ldots, x_{i-1} \in \{0,1\}$,

$$p_0 \leq \Pr[X_i = 1 | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \leq 1 - p_0.$$

A discouraging result of [SV86] is that there does not exist a deterministic extractor that extract a single bit from such sources. In other words, there are families of sources that are very structured and still do not allow deterministic extraction. This is bad news for the approach of simulating randomized algorithms

[3] The model considered by [vN51] is slightly different in that it places no bound on p. Instead, it allows the extractor to access an unbounded "stream" of independent coin tosses and requires that the extractor will stop and output completely uniform bits in expected time that depends on p. Nevertheless, the same algorithm applies in both cases.

and protocols with weak sources by deterministic extraction. Historically, this led to the notion of seeded extractors that we describe in Section 3. Nevertheless, as we discuss below, deterministic extraction has many applications beyond the original motivation of Section 1.

2.5 Bit-Fixing Sources and Privacy Amplification

We now consider the family of bit-fixing sources. In such a source some bits are independent unbiased coin tosses, while others are constants.

Definition 6 (bit-fixing sources). *A distribution X over $\{0,1\}^n$ is a k-bit-fixing source if there exist k distinct indices $i_1, \ldots, i_k$ such that the distribution $(X_{i_1}, \ldots, X_{i_k})$ is distributed like U_k and for $i \notin \{i_1, \ldots, i_k\}$, X_i is a fixed constant.*[4]

Bit-fixing sources do not seem to arise naturally in the scenario considered in Section 1. Nevertheless, as we observe now, they arise naturally in Cryptography [CGH+85]. Consider for example the following scenario. Assume that two parties Alice and Bob share a uniformly chosen random key $K \in \{0,1\}^n$. The key K is *private* in the sense that eavesdropper Eve has no information about it. Alice and Bob can securely encrypt communication between them using a shared private key. Suppose that at some stage, the privacy of K is somehow compromised and Eve learns $f(K)$ where $f(x) = x_{j_1}, \ldots, j_{n/2}$ for some indices $j_1, \ldots, j_{n/2}$ that are *unknown* to Alice and Bob. Alice and Bob would like to recover their privacy and come up with a new shared private key K'.

This can be achieved as follows: Let $E : \{0,1\}^n \to \{0,1\}^m$ be an ϵ-extractor for $(n/2)$-bit-fixing sources. Each party can compute $K' = E(K)$. We claim that K' is private in the sense that the distribution $(K', f(K))$ is ϵ-close to the distribution $(Z, f(K))$ where Z is uniformly distributed and independent of K. In order to see the meaning of thus claim, note that if $\epsilon = 0$ then the claim says that K' is independent of $f(K)$. For general $\epsilon > 0$ the claim implies that even an all-powerful Eve cannot distinguish K' from uniform with advantage greater than ϵ, given her view.

In order to prove the claim we note that for any value $v \in \mathsf{Supp}(f(K))$, the distribution $X = (K|f(K) = v))$ is an $(n/2)$-bit-fixing source. X captures the view that Eve has on K conditioned on the event $\{f(K) = v\}$. It follows that $E(X) = (E(K)|f(K) = v)$ is ϵ-close to uniform (and this holds for every $v \in \mathsf{Supp}(f(K))$. If $\epsilon = 0$ then this implies that $K' = E(K)$ is uniformly distributed and independent of $f(K)$. For general $\epsilon > 0$, this implies the statement in the claim above. (We remark that this argument can be generalized for other choices of "allowed functions" f as explained in Section 2.6.

Note that this application requires E to be explicit. Furthermore, Alice and Bob would like the new key to be as long as possible and this motivates extractors

[4] We remark that in the literature on extractors these sources are sometimes referred to as "oblivious bit-fixing sources" to distinguish them from "non-oblivious bit-fixing sources" in which for $i \notin \{i_1, \ldots, i_k\}$, X_i is some arbitrary function of $X_{i_1}, \ldots, X_{i_k}$.

that extract as many bits as possible from the k random bits that are "present" in a k-bit-fixing source.

It is trivial that the function $E(x) = (\sum_{1 \leq i \leq n} x_i) \bmod 2$ is a 0-extractor for k-bit-fixing sources for every $k \geq 1$. This simple idea does not easily extend to extract many bits for general k, as it was shown in [CGH+85] that there are no 0-extractors with $m > 1$ for $k < n/3$. The problem of extracting many bits was considered in [CGH+85, KZ07, GRS06, Rao09b] and the current state of the art is that there are explicit extractors that extract $m = (1 - o(1))k$ random bits from k-bit-fixing sources for every $k \geq \text{polylog}(n)$.

2.6 Families of Sources in the Literature on Deterministic Extraction

The literature on deterministic extractors and dispersers considers many families of sources. We briefly survey some of these families below. Our emphasis in the discussion below is on the min-entropy threshold parameter k. This is partly because there are general techniques to increase the output length of deterministic extractors [Sha08] and dispersers [GS08]. Using these techniques, it is often the case that extractors and dispersers that extract $\Omega(\log n)$ bits can be transformed into ones that extract $(1 - o(1)) \cdot k$ bits.

Multiple independent sources: Let $n = \ell \cdot n'$ and identify $\{0,1\}^n$ with $(\{0,1\}^{n'})^\ell$. A distribution $X = (X_1, \ldots, X_\ell)$ is an ℓ-independent source if $X_1, \ldots, X_\ell$ are independent. There exist extractors for 2-independent sources assuming $H_\infty(X_1), H_\infty(X_2) \geq \Omega(\log n)$. Most of the literature on deterministic extractors and dispersers focuses on multiple independent sources. We focus on this setup in Section 4.

Affine sources: Let $\mathbb{F}_q$ be the finite field with q elements. Affine sources are distributions that are uniform over some affine subspace of the vector space $\mathbb{F}_q^n$. The min-entropy of such sources coincides with the dimension of the affine space. Most of the research on explicit constructions focuses on the case $q = 2$ [BKS+10, Bou07, Rao09b, BSK09, Yeh10, Li11b, Sha11]. Explicit constructions of extractors and dispersers for affine sources are far from approaching the existential bounds proven using the probabilistic method. The latter show existence of extractors for affine sources with min-entropy at least $k = O(\log n)$. The best known explicit extractor is due to Bourgain [Bou07] and works for affine sources with min-entropy at least $k = o(n)$ (slight improvements to $k = n/\sqrt{\log \log n}$ were given in [Yeh10, Li11b]). It is possible to do better for dispersers and Shaltiel [Sha11] constructs an explicit disperser for affine sources with min-entropy at least $k = n^{o(1)}$. Explicit constructions do much better for "large fields" in which $q = n^{\Theta(1)}$ [GR08, GS08, DG10] and are able to extract from affine sources with min-entropy $O(\log n)$.

Feasibly generated sources: This approach considers families of sources that are specified by placing limitations on the process that generates the source. It was initiated by Blum [Blu86] that considered sources generated by finite Markov chains (see also [Vaz87]). A computational perspective was suggested

by Trevisan and Vadhan [TV00] (see also [KM05, KRVZ11]) who consider sources $X = C(U_r)$ where $C : \{0,1\}^r \to \{0,1\}^n$ is a "computational device". Different families of sources are obtained by placing limitations on the complexity of C. Note that affine sources are captured if C is a degree one polynomial. It is also natural to consider polynomials with larger degree [DGW09].

Feasibly recognizable sources: This approach (explicitly suggested in [Sha09]) considers sources that are uniform over sets of the form $\{x : f(x) = v\}$ for functions f coming from some specified class. Note that bit-fixing sources are captured by considering functions that are projections and affine sources are captured (once again) by considering degree one polynomials. It is also natural to consider polynomials with larger degrees [Dvi09]. Other families of sources can also be captured thus way: Sources recognizable by decision trees are convex combinations of bit-fixing sources and sources recognizable by 2-party communication protocols are convex combinations of 2-independent sources. (This is useful as an extractor for some family $\mathcal{C}$ is also an extractor for convex combinations of sources from the family). We also remark that the argument of Section 2.5 showing that Alice and Bob can recover their privacy when the function f is a projection, immediately extends to any class of functions f if one can explicitly construct an extractor for distributions recognizable by the class. Other applications of such extractors are given in [Sha09, KvMS09].

3 Seeded Extractors

These are extractors that in addition to one sample from the source distribution X also receive a second input Y (called "seed") which consists of (few) independent truly random bits.

Definition 7 (seeded extractors). *[NZ96] A function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a* (k,ϵ)-extractor *if for every distribution X over $\{0,1\}^n$ with $H_\infty(X) \geq k$, $E(X,Y)$ is ϵ-close to uniform (where Y is distributed like U_d and is independent of X). E is a* (k,ϵ)-disperser *if $|\mathsf{Supp}(E(X,Y))| \geq (1-\epsilon)\cdot 2^m$. E is a* strong extractor *if $E'(x,y) = (E(x,y),y)$ is an extractor.*

The definition above does not consider specific families $\mathcal{C}$ of sources. This is because seeded extractors are able to use a logarithmic length seed to extract from the "maximal family" of all distributions with "large" min-entropy.

In this section we survey some of the research on seeded extractors. In Section 3.1 we discuss explicit constructions and lower bounds. In Section 3.2 we explain how seeded extractors can be used to efficiently simulate randomized algorithms with access to a weak source. It turns out that seeded extractors have many other applications beyond the original motivation discussed in Section 1. We discuss some of these applications below. In Section 3.3 we point out a useful connection between seeded extractors and list-deocdable error-correcting codes. In Section 3.4 we interpret seeded extractors as graphs with large "relative expansion". In

Section 3.5 we show that seeded extractors can be used to construct graphs with expansion beating eigenvalue bounds. In Section 3.6 we point out that seeded extractors yield optimal averaging samplers.

3.1 Explicit Constructions and Lower Bounds

By the probabilistic method it is not hard to show that for every n, k, ϵ there exist (k, ϵ)-extractors that use a seed of length $d = \log(n-k) + 2\log(1/\epsilon) + O(1)$, and output $m = k + d - 2\log(1/\epsilon) - O(1)$ bits. There are lower bounds of Radhakrishnan and Ta-Shma [RTS00] showing that this is optimal (except for the constants hidden in the $O(1)$).

The quantity $k + d - m$ is referred to as the "entropy loss" of the extractor (as the input distribution (X, Y) has min-entropy $k + d$). It is obvious that the entropy loss is always non-negative. The lower bounds of [RTS00] stated above show that the entropy loss is at least $2\log(1/\epsilon) - O(1)$. This means that as ϵ decreases, some randomness must be lost in the extraction process.

A long line of research attempts to match the parameters of the existential bounds with explicit constructions.[5] There are explicit constructions that achieve:

Extractors optimal up to constant factors: For every constant $\alpha > 0$ there exists a constant c such that for every n, k, ϵ there are explicit extractors with seed length $d = c \cdot (\log n + \log(1/\epsilon))$, output length $m = (1-\alpha)k$. This was achieved by Lu et al. [LRVW03] for constant error ϵ and by Guruswami, Umans and Vadhan and [GUV09] for general error.

Extractors with sublinear entropy loss for large error: For every constant e there is a constant c such that for every n, k there are explicit extractors with seed length $d = c \cdot \log n$ and output length $m = (1 - \frac{1}{\log^e n}) \cdot k$ and error $\epsilon = 1/\log n$. This was achieved by Dvir et al. [DKSS09].

We remark that it is sometimes possible to do better for specific values of k, and that there are constructions that can push the leading constant c to $1 + o(1)$ while paying a little bit in some of the other parameters. The reader is referred to a survey article on explicit constructions of seeded extractors [Sha02].

3.2 Simulating BPP with Access to a Weak Random Source

Unlike deterministic extractors, seeded extractors expect to receive a short seed of truly random bits. Such a seed is not available in the scenario described in Section 1. Nevertheless, we now explain how to simulate any polynomial time randomized algorithm in polynomial time when given access to a general weak random source with sufficient min-entropy.

[5] Explicit seeded extractors are defined in a similar fashion to deterministic extractors: Let $d(\cdot), k(\cdot), m(\cdot)$ and $\epsilon(\cdot)$ be functions over integers. A function E that takes two strings and returns a string is an explicit $(k(\cdot), \epsilon(\cdot))$-extractor if for every sufficiently large n, when the first input x is of length n, E uses seed length $d(n)$, has output length $m(n)$ and is a $(k(n), \epsilon(n))$-extractor.

Let A be a randomized algorithm that runs in polynomial time and solves some decision problem with error $\leq 1/3$ (meaning that for every input, the probability that A answers incorrectly is at most $1/3$ where the probability is over choosing random coins for A). Assume that on an input x' of length n', A requires $m = \text{poly}(n')$ truly random bits. Assume that we can sample from some unknown distribution X over $\{0,1\}^n$ where $n = \text{poly}(n')$ with the guarantee that $H_\infty(X) \geq k$ for $k \geq 2m$. Let $E : \{0,1\}^n \times \{0,1\}^{O(\log n)} \to \{0,1\}^m$ be an explicit $(k, \frac{1}{10})$-extractor (that exists by the discussion in Section 3.1).

We simulate A as follows: When given input $x' \in \{0,1\}^{n'}$ and a sample x from the source X, for every seed y of E, we compute a bit v_y by applying A on input x' using $E(x,y)$ as a sequence of "random coins". The final output is the value v such that $v_y = v$ for most seeds y. It is not hard to see that for every input x' this process solves the decision problem with error less than $1/3 + 1/10 < 1/2$ where the probability is over the choice of x from X.[6]

Note that for this application it is crucial that E is explicit. Moreover, the simulation described above goes over all 2^d seeds of the extractor E. Consequently, it is crucial that $d = c \log n$ for some constant c to obtain a polynomial time simulation. Furthermore, the constant c determines the exponent of the polynomial (which gives motivation for constructing extractors with a small leading constant c) [TSZS06, SU05].

Inapplicability of this approach in cryptographic or distributed settings. It is important to stress that while this approach works for simulating randomized algorithms, it is not applicable when simulating randomized protocols in cryptographic or distributed settings. This is because the approach sketched above requires running the original protocol $2^d = n^{O(1)}$ times with many sequences of random coins. In cryptographic settings this means that adversaries get to participate in interactions in which the key is not necessarily random (which compromises the security of the protocol). In distributed settings, the total overhead incurred in running the initial protocol $n^{O(1)}$ times, leads to protocols that are inefficient and uninteresting. In Section 4.1 we suggest an alternative approach to simulate randomized protocols given access to multiple independent sources.

3.3 Seeded Extractors and List-Decodable Error-Correcting Codes

List-decodable error-correcting codes have many applications in computer science and are extensively studied. The reader is referred to [Gur07] for a survey articles on this notion and its applications. The definition below uses a nonstandard choice of letters preparing for the application below.

Definition 8 (List-decodable code). *For $x, y \in \{0,1\}^n$, let $\delta(x,y)$ denote the relative Hamming distance of x and y, that is $\delta(x,y) = \frac{|\{i : x_i \neq y_i\}|}{n}$. A function*

[6] In fact, the analysis can be improved and show that the error probability is bounded by $2^{-\Omega(k)}$ if we set the extractor to extract from distributions with min-entropy k' for $m \leq k' \leq \alpha k$ for a constant $\alpha < 1$. This is because of the connection between extractors and averaging samplers that we explain later on in Section 3.6.

$C : \{0,1\}^n \to \{0,1\}^{2^d}$ *is an* (ℓ, ϵ)*-list-decodable code if for every* $z \in \{0,1\}^{2^d}$, $\left|\left\{x : \delta(C(x), z) \leq \frac{1}{2} - \epsilon\right\}\right| \leq \ell$.

The definition above says that if one encodes a string $x \in \{0,1\}^n$ by $C(x)$ and then transmits $C(x)$ on a "noisy channel" that is allowed to adversarially choose $1/2-\epsilon$ of the indices of $C(x)$ and flip them to obtain a string z, then the receiver (who only sees z) knows a list of at most ℓ messages, such that one of them is the original message x. (The more standard notion of uniquely decodable codes is captured by the special case where $\ell = 1$ and it is known that such codes can only exist for $\epsilon > 1/4$.)

It turns out that strong seeded extractors that extract a single bit are essentially equivalent to list-decodable error correcting codes using the translation $E(x,y) = C(x)_y$ with $k \approx \log \ell$. This was first observed by Trevisan [Tre01]. A precise statement is given below:

- If $C : \{0,1\}^n \to \{0,1\}^{2^d}$ is an (ℓ, ϵ)-error correcting code then $E(x,y) = C(x)_y$ is a strong $(k, 2\epsilon)$-extractor for $k = \log \ell + \log(1/\epsilon) + 1$.
- If $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}$ is a strong (k, ϵ)-extractor then $C(x)$ defined by $C(x)_y = E(x,y)$ is a $(2^k - 1, 2\epsilon)$-list decodable code.

Proof. (sketch) For the second item, we note that if C is not a list-deocdable code, then there exists z which violates Definition 8 and has a list of size at least 2^k. The extractor E can be shown to fail on the distribution X that is uniform on this set. This is because for every $x \in \mathsf{Supp}(X)$, the "predictor function" $p(y) = z_y$ has $\Pr[p(Y) = E(x,Y)] \geq 1/2 + 2\epsilon > 1/2 + \epsilon$. It follows that $\Pr[p(Y) = E(X,Y)] > 1/2 + \epsilon$ which is a contradiction to the correctness of the extractor.

For the first item we note that if E is not a strong extractor, then there exist a source X and an event A such that $\Pr[(E(X,Y),Y) \in A]$ and $\Pr[U_{d+1} \in A]$ differ by more than 2ϵ. By standard arguments, such an event gives rise to a "predictor function" $p : \{0,1\}^d \to \{0,1\}$ that has $\Pr[p(Y) = E(X,Y)] > 1/2 + 2\epsilon$. (Intuitively, this follows because Y is uniformly distributed and so A does not gain from trying to distinguish Y from uniform, and has to be able to predict $E(X,Y)$ when given Y). By an averaging argument, there exist a set S consisting of an ϵ fraction of $x \in \mathsf{Supp}(X)$ such that for every $x \in S$ we have $\Pr[p(Y) = E(x,Y)] > 1/2 + \epsilon$. Function p gives rise to a string $z \in \{0,1\}^{2^d}$ by $z = p(y)_{(y \in \{0,1\}^d)}$ which contradicts Definition 8 as any message in S belongs to the list of z. □

The translation above can be used to present a *unified theory* of extractors, error-correcting codes (as well as other objects such as hash functions and expander graphs). The reader is referred to [Vad07] for such a treatment.

Exploiting this connection in explicit constructions. The connection above immediately gives excellent explicit constructions of strong extractors that extract one bit (by using known list-decodable codes). In order to extract many bits it is natural to have $E(x,y) = C(x)_{y_1}, \ldots, C(x)_{y_m}$ for some mapping $y \to y_1, \ldots, y_m$.

This approach is used by many constructions starting with Trevisan's breakthrough construction [Tre01] (see also [RRV02]) in which the mapping used is the Nisan-Wigderson pseudorandom generator [NW94]. A different mapping is used by Shaltiel and Umans [SU05] which also relies on a specific choice of the code. The aforementioned recent construction of Guruswami, Umans and Vadhan [GUV09] exploits this connection by using recent advances in coding theory (more specifically the Parvaresh-Vardy code [PV05]). The Parvaresh-Vardy code is not a code with Boolean alphabet, and so the translation above does not work directly. Nevertheless, a similar argument to the one given above can be used to construct "unbalanced expander graphs" which in turn yield seeded extractors.

A coding theoretic interpretation of extracting many bits. It was observed by Ta-Shma and Zuckerman [TSZ04] that strong extractors (k, ϵ)-extractors that extract $m > 1$ bits can be viewed (by the translation above) as codes over alphabet $\{0,1\}^m$ that allow list-decoding against channels that are extremely noisy in the following sense: When an encoding $C(x)$ of a message x passes through the channel, for every symbol $C(x)_y = E(x,y)$ of the encoding, the receiver gets a set $S_y \subseteq \{0,1\}^m$ of size say $2^m/2$ with the guarantee that $C(x)_y \in S_y$ for every y. "Extractor codes" allow recovery against such channels in the sense that the receiver knows a list of size 2^k of messages such that one of them is the original message x. In fact, extractor codes are resilient even against stronger channels that also "add errors" and are allowed adversarially choose $1/2 - \epsilon$ indices y in which S_y does not satisfy $C(x)_y \in S_y$.

3.4 Seeded Dispersers as Graphs with Expansion Properties

Given a function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ we set $N = 2^n$, $M = 2^m$, $D = 2^d$ and define a bipartite graph G_E where the left hand set of vertices is $\{0,1\}^n$, the right hand side set of vertices is $\{0,1\}^m$, and each vertex $x \in \{0,1\}^n$ is connected to $E(x,y)$ for every $y \in \{0,1\}^d$. Thus, the degree of vertices on the left hand side is D (and note that we allow multiple edges).

For a set $S \subseteq \{0,1\}^n$ on the left hand side, we define $\Gamma(S)$ to be the set of neighbors of S. It follows that if E is a (k,ϵ)-disperser (which follows in case E is a (k,ϵ)-extractor) then every set S of size at least $K = 2^k$ has $|\Gamma(S)| \geq (1-\epsilon)\cdot 2^m$. This notion resembles "vertex expansion" in so called "expander graphs". The reader is referred to [HLW06] for a manuscript on expander graphs and their many applications. We give a definition of vertex expansion below.

Definition 9 (Bipartite expander graphs). *A bipartite graph G is a (K,e)-expander if any set S on the left hand side of size at most K has $|\Gamma(S)| \geq e \cdot |S|$.*

In the definition above we consider bipartite graphs. This requirement is made to easily compare expander graphs with "disperser graphs". Note that standard expander graphs easily translate into bipartite ones. (Given a standard non-bipartite graph G in which every not too large set $S \subseteq V$ expands, we can create two copies of the vertices and imagine that edges go from the left-hand copy to the right-hand copy).

size vs. volume: In bipartite expander graphs sets expand in size in the sense that $|\Gamma(S)| \geq e \cdot |S|$. Disperser graphs may not expand in size and that all sets S have $|\Gamma(S)| < |S|$. Nevertheless, in disperser graphs, the set S expands in "volume" (the ratio of the size of the set and the size of the universe it lives in). More precisely, the volume of $S \subseteq \{0,1\}^n$ is $\frac{|S|}{N}$ (that may be very small and tend to zero), while the volume of $\Gamma(S) \subseteq \{0,1\}^m$ is $\frac{|\Gamma(S)|}{M} \geq (1-\epsilon)$.

Balanced vs. Unbalanced graphs: In many settings (and in particular in the setting of the application considered in Section 3.2) disperser graphs are "unbalanced", meaning that right had side which is of size M is much smaller than the left hand side which is of size N. Bipartite expander graphs are typically balanced in and the left hand side is of the same size as the right hand side. Nevertheless, it is interesting and useful to consider unbalanced bipartite expander graphs [TSUZ07, CRVW02, GUV09].

Constant degree vs. logarithmic degree: A useful property of expander graphs is that it is possible to have such graphs with constant degree. In contrast, the aforementioned lower bounds of [RTS00] imply that extractor graphs and disperser graphs must have degree at least $D \geq \Omega(\log(\frac{N}{K}))$ which is constant for $K = \Omega(N)$ and non-constant if $K = o(N)$. This means that disperser graphs cannot have constant degree if $K = o(N)$ (and the graph is unbalanced). Nevertheless, even bipartite expanders cannot have constant degree when the graph is sufficiently unbalanced. We perceive this observation as saying that the issue here is balanced vs. unbalanced rather than expander vs. disperser.

Sets smaller than K vs. sets larger than K: In expander graphs every set smaller than K expands, while in disperser graphs every set of size larger than K expands. This difference is a consequence of the difference between the notions of size and volume. In expander graphs, sets which are too large do not have "room" to expand in size, while in disperser graphs, sets which are too small cannot possibly expand to volume that is almost one unless the degree is huge.

There are many applications of extractor and disperser graphs in the literature. In some cases, extractors and dispersers give better performance than expander graphs. We present such examples in the next two sections.

3.5 Graphs with Expansion Beating the Eigenvalue Bound

We now present a result of Wigderson and Zuckerman [WZ99] showing that dispersers can be used to construct expanders with very strong expansion (for a particular parameter regime). We consider the following problem: Let $A \leq N/10$ be a parameter. Design a graph with small degree on N vertices such that every two sets of N/A vertices have an edge between them. (This is indeed a form of vertex expansion as it means that every set of size N/A sees more than $N - N/A \geq \frac{9N}{10}$ vertices).

Obviously, it is impossible to achieve this property with degree $o(A)$. The probabilistic method shows existence of such graphs with degree $\approx A \cdot \log(A)$. If

one achieves vertex expansion by analyzing the "spectral gap" of the graph, then the best possible degree is $\approx A^2$ in the sense that analysis of the spectral gap gives degree $\approx A^2$, and there exist graphs with optimal spectral gap in which the degree is $\approx A^2$ [Kah06].

Using optimal dispersers it is possible to almost match the probabilistic method and obtain degree $A \cdot \text{polylog}(A)$ (and such bounds can be approached using explicit constructions of dispersers for various choices of A [RVW00, SSZ98, TS02]).

The construction of [WZ99] works as follows: Let $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a $(k, 1/4)$-disperser for $k = \log(N/A)$ and $m = k$. (Furthermore, we need to assume that all nodes on the right hand side of the disperser graph have degree not much larger than the average degree given by ND/M). Let S_1, S_2 be two sets of size $N/A = 2^k$. By the disperser property each of the two sets sees more than half the vertices on the right hand side. Therefore, S_1 and S_2 both see a common neighbor on the right hand side. Define a graph G on $\{0,1\}^n$ in which every two vertices are connected if they share a common neighbor in the disperser graph. By the previous argument every sets S_1, S_2 of size N/A have an edge between them. Moreover, the degree of the graph is given by $D \cdot \frac{ND}{M} = \frac{D^2 N}{M}$ which is indeed $A \cdot \text{polylog}(A)$ if we use a disperser graph with small degree $D = \text{polylog}(N/K)$ (or equivalently short seed $d = O(\log(n-k))$).

Another example of graphs with expansion beating the eigenvalue bound is given by Capalbo et al. [CRVW02].

3.6 Seeded Extractors and Averaging Samplers

Let $Z_1, \ldots, Z_D$ be independent random variables over $\{0,1\}^m$, let $A \subseteq \{0,1\}^m$ and define indicator random variables $R_1, \ldots, R_d$ by $R_i = 1$ if $Z_i \in A$ and $R_i = 0$ otherwise. Let $Z = 2^{-D} \cdot \sum_{1 \le i \le D} Z_i$. The Chernoff bound gives that

$$\Pr[\left| Z - \frac{|A|}{2^m} \right| \ge \epsilon] \le \delta$$

for $\delta = 2^{-\Omega(\epsilon^2 D)}$. We say that random variables $Z_1, \ldots, Z_D$ (that are not necessarily independent) form an *averaging sampler* with estimation error ϵ and sampling error δ if they satisfy the inequality above for the parameters ϵ, δ.

Consider graphs over the vertex set $\{0,1\}^m$. A useful property of such graphs with "large spectral gap" is that if we choose a vertex Z_1 at random and take a random walk of length D on the graph to generate random variables $Z_1, \ldots, Z_D$ then we obtain random variables which form an averaging sampler (with quality that depends on the spectral gap) [AKS87, Gil98]. The advantage of this approach is that if the graph has constant degree then it is possible to generate the random walk variables $Z_1, \ldots, Z_D$ using only $m + O(D)$ random bits (compared to the $m \cdot D$ bits required to generate D independent random variables). This property has many applications in "derandomization theory" as it allows to approximate $|A|/2^m$ with small additive error using few random bits.

It was observed by Zuckerman [Zuc97] that seeded extractors yield averaging samplers with parameters that can beat those given by graphs with large spectral gap. The connection works as follows:

Extractors yield averaging samplers: Let $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a (k,ϵ)-extractor. Sample x uniformly from $\{0,1\}^n$, and set $Z_y = E(x,y)$. This gives random variables that are an averaging sampler with estimation error ϵ and sampling error $\delta = 2^{k+1}/2^n$. Moreover, sampling these variables requires n random bits.

Proof. (sketch) For every set $A \subseteq \{0,1\}^m$ we define

$$S_A = \left\{ x : \left| \Pr[E(x,Y) \in A] - \frac{|A|}{2^m} \right| > \epsilon \right\}$$

to be the set of strings with which the averaging sampler fails to estimate A correctly. To bound the size of S_A we note that w.l.o.g. at least half of its elements have the property above without the absolute value. The distribution X that is uniform over these elements is a source on which A distinguishes the output of the extractor from uniform. It follows that X does not have large min-entropy, which implies that S_A is small. □

It turns out that the connection between extractors and averaging samplers can also be reversed. Any procedure that uses n random bits to generate D random variables that form an averaging sampler with estimation error ϵ and sampling error δ, immediately translates into a $(k, 2\epsilon)$-extractor with $k = n - (\log(1/\delta) - \log(1/\epsilon))$ by setting $E(x,y) = Z_y$. Summing up, seeded extractors are essentially equivalent to averaging samplers under the translation $k \approx n - \log(1/\delta)$.

A consequence of this connection is that graphs with large spectral gap yield averaging samplers which in turn yield seeded extractors. This relationship is useful to construct extractors for very large k ($k \geq (1-\alpha)n$ for some constant $\alpha > 0$) but breaks down for smaller values.

4 Extractors for Multiple Independent Sources

Seeded extractors receive one source X (with the guarantee that $H_\infty(X) \geq k$ for some parameter k), and an independent short seed Y (that is uniformly distributed). The assumption that Y is uniformly distributed seems very strong, and we can try and relax it. A natural relaxation is to replace the requirement that Y is uniformly distributed by the weaker requirement that Y has large min-entropy. In this setup, it is no longer necessary to require that Y is short. A more natural setting of parameters is to have Y have the same length and min-entropy threshold as X. This leads to the notion of extractors for two independent sources. In fact, once we consider two independent sources, we may as well consider ℓ independent sources.

Definition 10 (Extractors and dispersers for multiple independent sources). *A function $E : (\{0,1\}^n)^\ell \to \{0,1\}^m$ is a (k,ϵ) ℓ-source extractor if for every ℓ independent random variables $X_1, \ldots, X_\ell$ such that for every $1 \leq i \leq \ell$, $H_\infty(X_i) \geq k$, it holds that $E(X_1, \ldots, X_\ell)$ is ϵ-close to U_m. E is a (k,ϵ) ℓ-source disperser if $|\mathsf{Supp}(E(X_1, \ldots, X_\ell))| \geq (1-\epsilon) \cdot 2^m$.*

We remark that these extractors are often seen as a special case of deterministic extraction (as explained in Section 2.6). We have seen in Section 2.1 that $(n-1,\epsilon)$ 1-source extractors/dispersers do not exist for $\epsilon < 1$. By the probabilistic method, 2-source extractors exist even for $k = O(\log n + \log(1/\epsilon))$.

In this section we survey some of the research on extractors and dispersers for multiple independent sources. In Section 4.1 we explain that multiple source extractors can be used to generate keys for cryptographic protocols. In Section 4.2 we show that explicit 2-source dispersers can be used to construct Ramsey graphs. In Section 4.3 we survey some of the explicit constructions extractors and dispersers for multiple independent sources.

4.1 Generating Keys for Cryptographic Protocols

In Section 3.2 we saw that it is possible to simulate randomized algorithms efficiently given access to *one* source. However, as explained there, that simulation is not applicable in cryptographic or distributed settings. Let us focus on cryptographic protocols. The security of such protocols depends on the ability of honest parties to generate uniformly distributed and private random keys. Moreover, in such settings the computer of an honest party may be operating in a "hostile environment" set up by an adversary that is trying to steal the secrets of the honest party.

ℓ-source extractors enable an honest party to sample a string that is (close to) uniform, assuming the party has access to ℓ independent sources. Each such source may be implemented by one of the approaches explained in Section 1. The requirement from each individual weak source is minimal: It should contain some min-entropy. It is plausible to assume that samples taken from sources that are "unrelated" or "remote" are independent. If we accept this assumption then we can generate keys for cryptographic protocols. Moreover, by the aforementioned discussion we can hope to have $\ell = 2$ and we only require independence between two random variables.

On a philosophical level the ability to generate independent random variables is a pre-requisite of cryptography. This is because a world in which this is not possible is a world in which there are no secrets (as every secret that we generate is correlated with other random variables that may become known to the adversary when we interact with him).

4.2 2-Source Dispersers and Ramsey Graphs

An (undirected) graph G on N vertices is K-Ramsey if there are no cliques or independent sets of size K in G. In 1947 (in the paper that introduced the celebrated probabilistic method) Erdös showed the existence of K-Ramsey graphs for $K \approx \log N$. It is a longstanding challenge to explicitly construct such graphs. Until recently, the best results were achieved by the classical work of Frankl and Wilson [FW81] and matched by several other researchers [Alo98, Gro00] giving K-Ramsey graphs for $K \approx 2^{\sqrt{\log N}}$. Moreover, Gopalan [Gop06] showed that some of the techniques used to attack this problem cannot beat this bound.

We now observe that explicit errorless two-source dispersers that output one bit yield explicit constructions of Ramsey graphs.

2-source dispersers yield Ramsey graphs: An explicit $(k, 0)$ 2-source disperser $D : (\{0,1\}^n)^2 \to \{0,1\}$ translates into an explicit construction of 2^{k+1}-Ramsey graph on 2^n vertices.

In fact, 2-source dispersers yield bipartite-Ramsey graphs (which are even harder to construct than Ramsey graphs). We now explain this argument. Let $D : (\{0,1\}^n)^2 \to \{0,1\}$ be a $(k, 0)$ 2-source disperser. We can use D to define a bipartite graph $B = (L, R, E)$ by interpreting it as an adjacency matrix. More formally, the left hand set of vertices is $L = \{0,1\}^n$, the right hand set is $R = \{0,1\}^n$ and two nodes $v_1 \in L$, $v_2 \in R$ are connected iff $D(v_1, v_2) = 1$. Graph B is a balanced bipartite graph where each of the two sides has $N = 2^n$ vertices. Furthermore, G has the property that for every two sets $A, B \subseteq \{0,1\}^n$ of size $K = 2^k$, $D(A, B) = \{0,1\}$ meaning that every $K \times K$ induced subgraph of G cannot be empty nor complete. Such graphs are called "K-bipartite Ramsey graphs".

In particular, we have that for every set $A \subseteq \{0,1\}^n$ of size $K = 2^k$, $D(A, A) = \{0,1\}$. If D is symmetric (meaning that $D(x, y) = D(y, x)$) then we can also interpret it as the adjacency matrix of a (non-bipartite) undirected graph over vertex set $\{0,1\}^n$ and the property above means that D is a K-Ramsey graph (as every set A of size K is not a clique nor an independent set). We can make D symmetric by ordering the elements in $\{0,1\}^n$ in some arbitrary way and modifying $D(x, y)$ to $D(y, x)$ for $x > y$. (We can also force $D(x, x) = 0$ if we want to avoid self loops). This modification does not spoil D by much. It is easy to see that if D is a $(k, 0)$ 2-source disperser than following the modification it is still a $(k + 1, 0)$ 2-source disperser. Summing up, when given disperser D we can symmetrize it and use it to define the adjacency matrix of a Ramsey graph.

4.3 Explicit Constructions of ℓ-Source Extractors and Dispersers

2-source extractors and dispersers. The discussion above explains some of the difficulty in constructing explicit 2-source extractors and dispersers for small k. We now survey the known constructions. Chor and Goldreich [CG88] showed that $E(x, y) = (\sum_{1 \le i \le n} x_i \cdot y_i) \bmod 2$ is a 2-source extractor with very small error if k is sufficiently larger than $n/2$. (Earlier work by Santha and Vazirani [SV86] considered two independent Santha-Vazirani sources and analyzed the same extractor function). Bourgain [Bou05] (see also [Rao07]) improved the entropy threshold to $k = (1/2 - \alpha)n$ for some small constant $\alpha > 0$. (The seemingly small difference between $n/2$ and $(1/2 - \alpha)n$ plays a crucial role in some of the recent developments in this area). This is the best known extractor construction for two sources with the same min-entropy. Raz [Raz05] constructed 2-source extractors where one source has min-entropy larger than $n/2$ and the other has logarithmic entropy. Shaltiel [Sha08] used Raz's extractor to give a 2-source extractor which for two sources with min-entropy $k > n/2$ extracts $(2 - o(1)) \cdot k$ random bits out of the $2k$ bits of entropy that are available in the two sources.

Barak et al. [BKS+10] constructed 2-source dispersers that for every $\delta > 0$ achieve $k = \delta n$. Barak et al. [BRSW06] extended the technique of [BKS+10] and constructed 2-source dispersers for $k = 2^{\log^{0.9} n} = n^{o(1)}$. By the discussion in Section 4.2 such dispersers translate into Ramsey graphs that beat the Frankl-Wilson construction [FW81] and give the state of the art on this problem. The constructions of [BKS+10, BRSW06] are quite involved and rely on many components from the extractor literature.

ℓ-source extractors. Barak, Impagliazzo and Wigderson [BIW06] constructed extractors that for every $\delta > 0$, use $\text{poly}(1/\delta)$ sources with min-entropy $k = \delta n$. The key was exploiting recent developments in arithmetic combinatorics (in particular, the "sum-product theorem" of [BKT04, Kon03]) to analyze a construction that was previously suggested by Zuckerman. The high level idea is to show that if X, Y, Z are three independent sources with min-entropy δn, where each source is over a finite field $\mathbb{F}$ that has no large subfields (which holds vacuously if $\mathbb{F}$ is a prime field) then $X \cdot Y + Z$ is a distribution that is (close to) having min-entropy $min((\delta + \alpha)n, n)$ for some constant $\alpha > 0$. By iteratively increasing the entropy, this gives an extractor.

Rao [Rao09a] used machinery from seeded extractors to construct extractors that extract randomness from $O(\frac{\log n}{\log k})$ independent sources with min-entropy k. Note that this means that only $O(1/\delta)$ sources are needed for $k = n^{\delta}$. Rao starts by observing that every source X over $\{0,1\}^n$ with $H_\infty(X) \geq k$ can be transformed into a "somewhere-random" source X' consisting of $n^{O(1)}$ blocks of length $\Omega(k)$ where at least one of them is (close to) uniformly distributed. (Each such block B_y is obtained by $B_y = E(X, y)$ where E is a (k, ϵ)-seeded extractor with seed length $O(\log n)$ so that there are $n^{O(1)}$ blocks). This reduces the task of extracting from independent general sources to that of extracting from independent somewhere random sources, and this turns out to be easier.

In recent years there is ongoing research that aims to improve the results above.

5 Some Open Problems

Below we state some open problems related to explicit constructions of extractors. We remark that there are many other open problems that are related to applications of extractors.

Deterministic extractors.

- Construct affine extractors for the field $\mathbb{F}_2$ with min-entropy $k < \sqrt{n}$. The best known construction achieves $k = n/\sqrt{\log \log n}$ [Bou07, Yeh10, Li11b].
- Construct affine dispersers for the field $\mathbb{F}_2$ with min-entropy $k = \text{polylog}(n)$. The best known construction achieves $k = 2^{\log^{0.9} n}$ [Sha11].
- For every constant c, construct extractors for sources samplable by circuits of size n^c with min-entropy $k < n/2$. It is allowed to use any "plausible hardness assumption". A construction based on worst-case hardness against a strong variant of nondeterministic circuits was given in [TV00]. This construction requires $k = (1 - \alpha)n$ for some constant $\alpha > 0$.

Seeded extractors.

– Construct seeded extractors that match the lower bounds of [RTS00]. These lower bounds and state of the art are described in Section 3.1 (see also [Sha02]).

Multiple source extractors.

– Construct 2-source extractors for min-entropy $k \leq n/3$. The best known construction achieves $k = (1/2 - \alpha)n$ for some constant $\alpha > 0$ [Bou05].
– Construct 2-source dispersers for min-entropy $k = \text{polylog}(n)$. The best known construction achieves $k = 2^{\log^{0.9} n}$ [BRSW06].
– Construct ℓ-source extractors for $\ell = O(1)$ and min-entropy $k = \text{polylog}(n)$. The best known construction achieves $\ell = O(\frac{\log n}{\log k})$ which is constant only for $k = n^{\Omega(1)}$ [Rao09a].

References

[AB09] Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, Cambridge (2009)

[AKS87] Ajtai, M., Komlós, J., Szemerédi, E.: Deterministic simulation in logspace. In: STOC, pp. 132–140 (1987)

[Alo98] Alon, N.: The shannon capacity of a union. Combinatorica 18(3), 301–310 (1998)

[BIW06] Barak, B., Impagliazzo, R., Wigderson, A.: Extracting randomness using few independent sources. SIAM J. Comput. 36(4), 1095–1118 (2006)

[BKS+10] Barak, B., Kindler, G., Shaltiel, R., Sudakov, B., Wigderson, A.: Simulating independence: New constructions of condensers, ramsey graphs, dispersers, and extractors. J. ACM 57(4) (2010)

[BKT04] Bourgain, J., Katz, N., Tao, T.: A sum-product estimate in finite fields, and applications. Geom. Funct. Anal. 14(1), 27–57 (2004)

[Blu86] Blum, M.: Independent unbiased coin flips from a correlated biased source-a finite stae markov chain. Combinatorica 6(2), 97–108 (1986)

[Bou05] Bourgain, J.: More on the sum-product phenomenon in prime fields and its applications. International Journal of Number Theory 1, 1–32 (2005)

[Bou07] Bourgain, J.: On the construction of affine extractors. Geometric And Functional Analysis 17(1), 33–57 (2007)

[BRSW06] Barak, B., Rao, A., Shaltiel, R., Wigderson, A.: 2-source dispersers for sub-polynomial entropy and ramsey graphs beating the frankl-wilson construction. In: STOC, pp. 671–680 (2006)

[BSK09] Ben-Sasson, E., Kopparty, S.: Affine dispersers from subspace polynomials. In: STOC, pp. 65–74 (2009)

[BSZ11] Ben-Sasson, E., Zewi, N.: From affine to two-source extractors via approximate duality. In: STOC (2011)

[CG88] Chor, B., Goldreich, O.: Unbiased bits from sources of weak randomness and probabilistic communication complexity. SIAM Journal on Computing 17(2), 230–261 (1988)

[CGH+85] Chor, B., Goldreich, O., Håstad, J., Friedman, J., Rudich, S., Smolensky, R.: The bit extraction problem of t-resilient functions (preliminary version). In: FOCS, pp. 396–407 (1985)

[CRVW02] Capalbo, M.R., Reingold, O., Vadhan, S.P., Wigderson, A.: Randomness conductors and constant-degree lossless expanders. In: STOC, pp. 659–668 (2002)

[DG10] DeVos, M., Gabizon, A.: Simple affine extractors using dimension expansion. In: IEEE Conference on Computational Complexity, pp. 50–57 (2010)

[DGW09] Dvir, Z., Gabizon, A., Wigderson, A.: Extractors and rank extractors for polynomial sources. Computational Complexity 18(1), 1–58 (2009)

[DKSS09] Dvir, Z., Kopparty, S., Saraf, S., Sudan, M.: Extensions to the method of multiplicities, with applications to kakeya sets and mergers. In: FOCS, pp. 181–190 (2009)

[Dvi09] Dvir, Z.: Extractors for varieties. In: IEEE Conference on Computational Complexity, pp. 102–113 (2009)

[Eli72] Elias, P.: The efficient construction of an unbiased random sequence. Ann. Math. Statist. 43, 865–870 (1972)

[FW81] Frankl, P., Wilson, R.M.: Intersection theorems with geometric consequences. Combinatorica 1(4), 357–368 (1981)

[Gil98] Gillman, D.: A chernoff bound for random walks on expander graphs. SIAM J. Comput. 27(4), 1203–1220 (1998)

[Gop06] Gopalan, P.: Constructing ramsey graphs from boolean function representations. In: IEEE Conference on Computational Complexity, pp. 115–128 (2006)

[GR08] Gabizon, A., Raz, R.: Deterministic extractors for affine sources over large fields. Combinatorica 28(4), 415–440 (2008)

[Gro00] Grolmusz, V.: Superpolynomial size set-systems with restricted intersections mod 6 and explicit ramsey graphs. Combinatorica 20(1), 71–86 (2000)

[GRS06] Gabizon, A., Raz, R., Shaltiel, R.: Deterministic extractors for bit-fixing sources by obtaining an independent seed. SIAM J. Comput. 36(4), 1072–1094 (2006)

[GS08] Gabizon, A., Shaltiel, R.: Increasing the output length of zero-error dispersers. In: APPROX-RANDOM, pp. 430–443 (2008)

[Gur07] Guruswami, V.: Algorithmic results in list decoding. Foundations and Trends in Theoretical Computer Science 2(2) (2007)

[GUV09] Guruswami, V., Umans, C., Vadhan, S.P.: Unbalanced expanders and randomness extractors from parvaresh–vardy codes. J. ACM 56(4) (2009)

[HLW06] Hoory, S., Linial, N., Wigderson, A.: Expander graphs and their applications. Bull. Amer. Math. Soc. 43, 439–561 (2006)

[Kah06] Kahale, N.: Eigenvalues and expansion of regular graphs. J. Assoc. Comput. Mach. 42, 1091–1106 (2006)

[KLR09] Kalai, Y.T., Li, X., Rao, A.: 2-source extractors under computational assumptions and cryptography with defective randomness. In: FOCS, pp. 617–626 (2009)

[KLRZ08] Kalai, Y.T., Li, X., Rao, A., Zuckerman, D.: Network extractor protocols. In: FOCS, pp. 654–663 (2008)

[KM05] König, R., Maurer, U.M.: Generalized strong extractors and deterministic privacy amplification. In: IMA Int. Conf., pp. 322–339 (2005)

[Kon03] Konyagin, S.V.: A sum-product estimate in fields of prime order. Arxiv technical report (2003), http://arxiv.org/abs/math.NT/0304217

[KRVZ11] Kamp, J., Rao, A., Vadhan, S.P., Zuckerman, D.: Deterministic extractors for small-space sources. J. Comput. Syst. Sci. 77(1), 191–220 (2011)
[KvMS09] Kinne, J., van Melkebeek, D., Shaltiel, R.: Pseudorandom generators and typically-correct derandomization. In: APPROX-RANDOM, pp. 574–587 (2009)
[KZ07] Kamp, J., Zuckerman, D.: Deterministic extractors for bit-fixing sources and exposure-resilient cryptography. SIAM J. Comput. 36(5), 1231–1247 (2007)
[Li11a] Li, X.: Improved constructions of three source extractors. In: IEEE Conference on Computational Complexity (2011)
[Li11b] Li, X.: A new approach to affine extractors and dispersers. In: IEEE Conference on Computational Complexity (2011)
[LRVW03] Lu, C.-J., Reingold, O., Vadhan, S.P., Wigderson, A.: Extractors: optimal up to constant factors. In: STOC, pp. 602–611 (2003)
[NTS99] Nisan, N., Ta-Shma, A.: Extracting randomness: A survey and new constructions. J. Comput. Syst. Sci. 58(1), 148–173 (1999)
[NW94] Nisan, N., Wigderson, A.: Hardness vs randomness. J. Comput. Syst. Sci. 49(2), 149–167 (1994)
[NZ96] Nisan, N., Zuckerman, D.: Randomness is linear in space. J. Comput. Syst. Sci. 52(1), 43–52 (1996)
[Per92] Peres, Y.: Iterating von neumann's procedure for extracting random bits. Ann. Statist. 20, 590–597 (1992)
[PV05] Parvaresh, F., Vardy, A.: Correcting errors beyond the guruswami-sudan radius in polynomial time. In: FOCS, pp. 285–294 (2005)
[Rao07] Rao, A.: An exposition of bourgain's 2-source extractor. Electronic Colloquium on Computational Complexity (ECCC) 14(034) (2007)
[Rao09a] Rao, A.: Extractors for a constant number of polynomially small min-entropy independent sources. SIAM J. Comput. 39(1), 168–194 (2009)
[Rao09b] Rao, A.: Extractors for low-weight affine sources. In: IEEE Conference on Computational Complexity, pp. 95–101 (2009)
[Raz05] Raz, R.: Extractors with weak random seeds. In: STOC, pp. 11–20 (2005)
[RRV02] Raz, R., Reingold, O., Vadhan, S.P.: Extracting all the randomness and reducing the error in trevisan's extractors. J. Comput. Syst. Sci. 65(1), 97–128 (2002)
[RTS00] Radhakrishnan, J., Ta-Shma, A.: Bounds for dispersers, extractors, and depth-two superconcentrators. SIAM Journal on Discrete Mathematics 13(1), 2–24 (2000)
[RVW00] Reingold, O., Vadhan, S.P., Wigderson, A.: Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In: FOCS, pp. 3–13 (2000)
[RZ08] Rao, A., Zuckerman, D.: Extractors for three uneven-length sources. In: APPROX-RANDOM, pp. 557–570 (2008)
[Sha02] Shaltiel, R.: Recent developments in explicit constructions of extractors. Bulletin of the EATCS 77, 67–95 (2002); Special issue on cryptography
[Sha08] Shaltiel, R.: How to get more mileage from randomness extractors. Random Struct. Algorithms 33(2), 157–186 (2008)
[Sha09] Shaltiel, R.: Weak derandomization of weak algorithms: Explicit versions of yao's lemma. In: IEEE Conference on Computational Complexity, pp. 114–125 (2009)
[Sha11] Shaltiel, R.: Dispersers for affine sources with sub-polynomial entropy (unpublished, 2011)

[SSZ98] Saks, M.E., Srinivasan, A., Zhou, S.: Explicit or-dispersers with polylogarithmic degree. J. ACM 45(1), 123–154 (1998)
[SU05] Shaltiel, R., Umans, C.: Simple extractors for all min-entropies and a new pseudorandom generator. J. ACM 52(2), 172–216 (2005)
[SV86] Santha, M., Vazirani, U.V.: Generating quasi-random sequences from semi-random sources. J. Comput. Syst. Sci. 33(1), 75–87 (1986)
[Tre01] Trevisan, L.: Extractors and pseudorandom generators. J. ACM 48(4), 860–879 (2001)
[TS02] Ta-Shma, A.: Almost optimal dispersers. Combinatorica 22(1), 123–145 (2002)
[TSUZ07] Ta-Shma, A., Umans, C., Zuckerman, D.: Lossless condensers, unbalanced expanders, and extractors. Combinatorica 27(2), 213–240 (2007)
[TSZ04] Ta-Shma, A., Zuckerman, D.: Extractor codes. IEEE Transactions on Information Theory 50(12), 3015–3025 (2004)
[TSZS06] Ta-Shma, A., Zuckerman, D., Safra, S.: Extractors from reed-muller codes. J. Comput. Syst. Sci. 72(5), 786–812 (2006)
[TV00] Trevisan, L., Vadhan, S.P.: Extracting randomness from samplable distributions. In: FOCS, pp. 32–42 (2000)
[Vad07] Vadhan, S.P.: The unified theory of pseudorandomness. SIGACT News 38(3), 39–54 (2007)
[Vaz87] Vazirani, U.V.: Efficiency considerations in using semi-random sources (extended abstract). In: STOC, pp. 160–168 (1987)
[vN51] von Neumann, J.: Various techniques used in connection with random digits. Applied Math Series 12, 36–38 (1951)
[VV85] Vazirani, U.V., Vazirani, V.V.: Random polynomial time is equal to slightly-random polynomial time. In: FOCS, pp. 417–428 (1985)
[WZ99] Wigderson, A., Zuckerman, D.: Expanders that beat the eigenvalue bound: Explicit construction and applications. Combinatorica 19(1), 125–138 (1999)
[Yeh10] Yehudayoff, A.: Affine extractors over prime fields (2010) (manuscript)
[Zuc97] Zuckerman, D.: Randomness-optimal oblivious sampling. Random Struct. Algorithms 11(4), 345–367 (1997)

Invitation to Algorithmic Uses of Inclusion–Exclusion

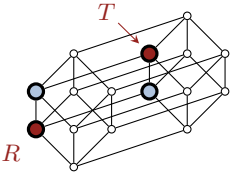
Thore Husfeldt

IT University of Copenhagen, Denmark
Lund University, Sweden

Abstract. I give an introduction to algorithmic uses of the principle of inclusion–exclusion. The presentation is intended to be be concrete and accessible, at the expense of generality and comprehensiveness.

1 The principle of inclusion–exclusion. There are as many odd-sized as even-sized subsets sandwiched between two different sets: For $R \subseteq T$,

$$\sum_{R \subseteq S \subseteq T} (-1)^{|T \setminus S|} = [R = T]\,. \tag{1}$$

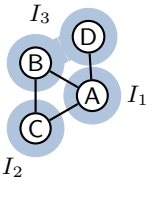


We use Iverson notation $[P]$ for proposition P, meaning $[P] = 1$ if P and $[P] = 0$ otherwise.

Proof of (1). If $R = T$ then there is exactly one sandwiched set, namely $S = T$. Otherwise we set up a bijection between the odd- and even-sized subsets as follows. Fix $t \in T \setminus R$. For every odd-sized subset S_1 with $R \subseteq S_1 \subseteq T$ let $S_0 = S_1 \oplus \{t\}$ denote the symmetric difference of S_1 with $\{t\}$. Note that the size of S_0 is even and that S_0 contains R. Furthermore, S_1 can be recovered from S_0 as $S_1 = S_0 \oplus \{t\}$. □

Perspective. We will see the (perhaps more familiar) formulation of the principle of inclusion–exclusion in terms of intersecting sets in §6, and another equivalent formulation in §11.

2 Graph colouring. A *k-colouring* of a graph $G = (N, E)$ on $n = |N|$ nodes assigns one of k colours to every node such that neighbouring nodes have different colours. In any such colouring, the nodes of the same colour form a nonempty *independent set*, a set of nodes none of which are neighbours.



Let $g(S)$ denote the number of nonempty independent subsets in $S \subseteq N$. Then G can be k-coloured if and only if

$$\sum_{S \subseteq N} (-1)^{n-|S|} \big(g(S)\big)^k > 0\,. \tag{2}$$

Base revision 41bba45…, Tue May 3 21:46:40 2011 +0200, Thore Husfeldt.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 42–59, 2011.

Proof. For every $S \subseteq N$, the term $g(S)^k$ counts the number of ways to pick k nonempty independent sets $I_1, \ldots, I_k$ in S. Thus, we can express the left hand side of (2) as

$$\sum_S \sum_{I_1} \cdots \sum_{I_k} [\forall i\colon I_i \subseteq S](-1)^{|N \setminus S|} = \sum_{I_1} \cdots \sum_{I_k} \sum_S [\forall i\colon I_i \subseteq S](-1)^{|N \setminus S|}.$$

The innermost sum has the form

$$\sum_{I_1 \cup \cdots \cup I_k \subseteq S \subseteq N} (-1)^{|N \setminus S|}.$$

By (1), the only contributions come from $I_1 \cup \cdots \cup I_k = N$. Every such choice indeed corresponds to a valid colouring: For $i = 1, \ldots, k$, let the nodes in I_i have colour i. (This may re-colour some nodes.) Conversely, every valid k-colouring corresponds to such a choice. (In fact, the colourings are the disjoint partitions). □

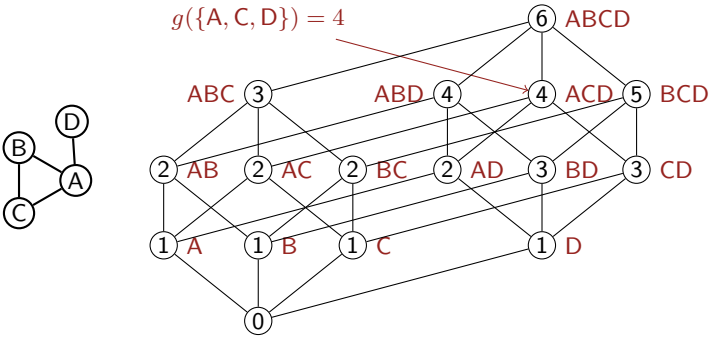


Fig. 1. The values of $g(S)$ for all S for the example graph to the left. Expression (2) evaluates an alternating sum of the cubes of these values, in this case $6^3 - (3^3 + 4^3 + 4^3 + 5^3) + (2^3 + 2^3 + 2^3 + 2^3 + 3^3 + 3^3) - (1^3 + 1^3 + 1^3 + 1^3) + 0 = 18$.

3 Counting the number of independent sets. Expression (2) can be evaluated in two ways:

For each $S \subseteq N$, the value $g(S)$ can be computed in time $O(2^{|S|}|E|)$ by constructing every nonempty subset of S and testing it for independence. Thus, the total running time for evaluating (2) is within a polynomial factor of

$$\sum_{S \subseteq N} 2^{|S|} = \sum_{i=1}^{n} \binom{n}{i} 2^i = 3^n.$$

The space requirement is polynomial.

Alternatively, we first build a table with 2^n entries containing $g(S)$ for all $S \subseteq N$, after which we can evaluate (2) in time and space $2^n n^{O(1)}$.

Such a table is easy to build given a recurrence for $g(S)$. We have $g(\varnothing) = 0$, and

$$g\big(S\big) = g\big(S \setminus \{v\}\big) + g\big(S \setminus N[v]\big) + 1 \qquad (v \in S)\,, \tag{3}$$

where $N[v] = \{v\} \cup \{\, u \in N \colon uv \in E \,\}$ denotes the closed neighbourhood of v.

Proof of (3). Fix $v \in S$ and consider the nonempty independent sets $I \subseteq S$. They can be partitioned into two classes: either $v \in I$ or $v \notin I$. The latter sets are counted in $g\big(S \setminus \{v\}\big)$. It remains to argue that the sets $I \ni v$ are counted in $g\big(S \setminus N[v]\big) + 1$. We will do this by counting the equipotent family of sets $I \setminus \{v\}$ instead. Since I contains v and is independent, it cannot contain other nodes in $N[v]$. Thus $I \setminus \{v\}$ is disjoint from $N[v]$ and contained in S. Now, either I is the singleton $\{v\}$ itself, accounted for by the '+1' term, or $I \setminus \{v\}$ is a nonempty independent set and therefore counted in $g\big(S \setminus N[v]\big)$. □

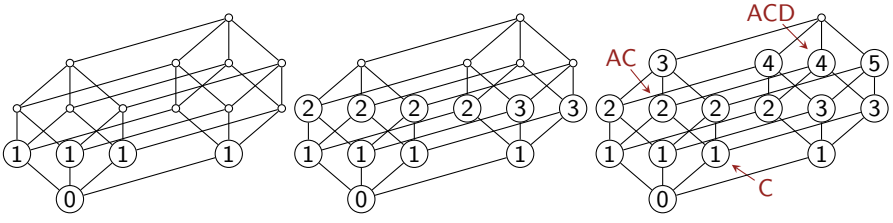


Fig. 2. Three stages in the tabulation of $g(S)$ for all $S \subseteq N$ bottom-up. For example, the value of $g(\{\mathsf{A}, \mathsf{C}, \mathsf{D}\})$ is given by (3) with $v = \mathsf{D}$ as $g(\{\mathsf{A}, \mathsf{C}\}) + g(\{\mathsf{C}\}) + 1 = 4$.

Perspective. The *brute force* solution for graph colouring tries all k^n assignments of colours to the nodes, which is slower for $k \geq 4$. Another approach is *dynamic programming over the subsets* [15], based on the idea that G can be k-coloured if and only if $G[N \setminus S]$ can be $(k-1)$-coloured for some nonempty independent set S. That algorithm also runs within a polynomial factor of 3^n, but uses exponential space. In summary, the inclusion–exclusion approach is faster than brute force, and uses less space than dynamic programming over the subsets. The insight that this idea applies to a wide range of sequencing and packing problems goes back to Karp [12], the application to graph colouring is from [2].

We use a space–time trade-off to reducing the exponential running time factor from 3^n to 2^n, applying dynamic programming to tabulate the decrease-and-conquer recurrence (3), based on [8]. Recurrence (3) depends heavily on the structure of independent sets; a more general approach is shown in §10.

The two strategies for computing $g(S)$ represent extreme cases of a space–time tradeoff that can be balanced [4].

4 Perfect matchings in bipartite graphs. Consider a bipartite graph with bipartition (N, N), where $N = \{1, \ldots, n\}$, and edge set $E \subseteq N \times N$. A *perfect matching* is an edge subset $M \subseteq E$ that includes every node as an endpoint exactly once. See Fig. 3 for some interpretations.

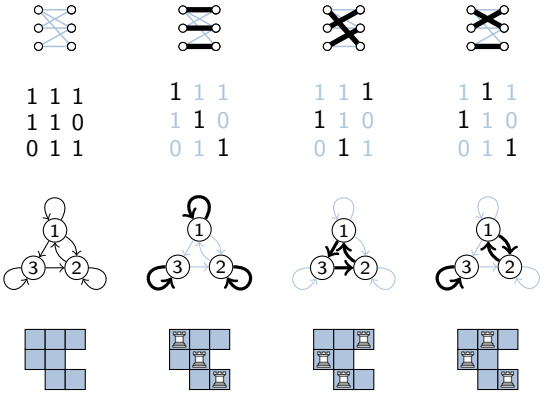


Fig. 3. Row 1: A bipartite graph and its three perfect matchings. Row 2: In the graph's adjacency matrix A, every perfect matching corresponds to a permutation π for which $A_{i,\pi(i)} = 1$ for all $i \in [n]$. Row 3: In the directed n-node graph defined by A, every perfect matching corresponds to a directed cycle partition. Bottom row: an equivalent formulation in terms of non-attacking rooks on a chess board with forbidden positions.

The *Ryser formula* for counting the perfect matchings in such a graph can be given as

$$\sum_{\pi \in S_n} \prod_{i=1}^{n} [i\pi(i) \in E] = \sum_{S \subseteq N} (-1)^{|N \setminus S|} \prod_{i=1}^{n} \sum_{j \in S} [ij \in E]\,, \tag{4}$$

where S_n denotes the set of permutations from N to N. The left hand side succinctly describes the problem as iterating over all permutations and checking if the corresponding edges (namely, $1\pi(1), 2\pi(2), \ldots, n\pi(n)$) are all in E. Direct evaluation would require $n!$ iterations. The right hand side provides an equivalent expression that can be evaluated in time $O(2^n n^2)$, see Fig. 4.

Proof of (4). For fixed $i \in N$, the value $\sum_{j \in S} [ij \in E]$ counts the number of i's neighbours in $S \subseteq N$. Thus the expression

$$\prod_{i=1}^{n} \sum_{j \in S} [ij \in E] \tag{5}$$

is the number of ways every node $i \in N$ can choose a neighbour from S. (This allows some nodes to select the same neighbour.) Consider such a choice as a mapping $g\colon N \to N$, not necessarily onto, with image $R = g(N)$. The contribution of g to (5) is 1 for every $S \supseteq R$, and its total contribution to the right hand side of (4) is, using (1),

$$\sum_{R \subseteq S \subseteq N} (-1)^{|N \setminus S|} \cdot 1 = [g(N) = N]\,.$$

Thus g contributes if and only if it is a permutation. □

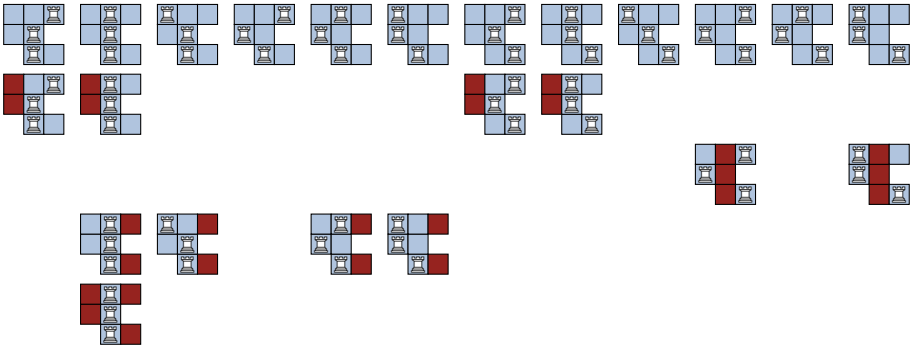


Fig. 4. Inclusion–exclusion for non-attacking rooks. The top row shows all $12 = 3 \cdot 2 \cdot 2$ ways to place exactly one rook in every board line. Every row shows the possible placements in the vertical lines given by $S \subseteq \{1, 2, 3\}$. We omit the rows whose contribution vanishes, namely $S = \{1\}$, $S = \{3\}$ and $S = \varnothing$. Of particular interest is the second column, which is subtracted twice and later added again. The entire calculation is $12 - 4 - 2 - 4 + 1 + 0 + 0 - 0 = 3$.

Perspective. Bipartite matching is an example of a sequencing problem, where inclusion–exclusion replaces an enumeration over permutations, $\sum_{\pi \in S_n}$ by an alternating enumeration over subsets $\sum_{S \subseteq N} (-1)^{|N \setminus S|}$ of functions with restricted range. Typically, this reduces a factor $n!$ in the running time to 2^n. One can express the idea algebraically like this:

$$\begin{aligned}
\sum_{\substack{f\colon N \to N \\ f(N)=N}} [\cdots] &= \sum_R [R = N] \sum_{\substack{f\colon N \to N \\ f(N)=R}} [\cdots] \\
&= \sum_R \sum_S [R \subseteq S](-1)^{|N \setminus S|} \sum_{\substack{f\colon N \to N \\ f(N)=R}} [\cdots] \\
&= \sum_S (-1)^{|N \setminus S|} \sum_R [R \subseteq S] \sum_{\substack{f\colon N \to N \\ f(N)=R}} [\cdots] \\
&= \sum_S (-1)^{|N \setminus S|} \sum_{f\colon N \to S} [\cdots]\,.
\end{aligned} \tag{6}$$

Ryser's formula is normally given in a more general form, for the *permanent* $\sum_\pi \prod_i A_{i\pi(i)}$ of a matrix, where the entries can be other than just 0 and 1. The running time can be improved to $O(2^n n)$ arithmetic operations by iterating over N in Gray code order.

Ryser's formula [17] is a very well-known result in combinatorics and appears in many textbooks. However, it is easy to achieve running time $O(2^n n)$ using dynamic programming over the subsets, at the expense of space $O(2^n)$. This is the standard approach to sequencing problems [1,11], and appears as an exercise in Knuth [13, pp. 515-516], but usually not in the combinatorics literature. We

will witness the opposite methodological preferences in §7. Inclusion–exclusion-based algorithms for the permanent of non-square matrices in semirings are described in [5].

5 Perfect matchings in general graphs. We turn to graphs that are not necessarily bipartite. In general, the number of perfect matchings in a graph with an even number n of nodes N is

$$\sum_{S\subseteq N} (-1)^{|N\setminus S|} \binom{e[S]}{n/2}, \tag{7}$$

where $e[S]$ denotes the number of edges between nodes in $S \subseteq N$.

Proof. The term $\binom{e[S]}{n/2}$ counts the number of ways to select $n/2$ distinct edges with endpoints in S. (The edges are distinct, but may share nodes.) Consider such a selection $F \subseteq E$ and let $R = \bigcup_{uv\in F}\{u, v\}$ denote the nodes covered by the selected edges. The total contribution of F to the right hand side of (7) is

$$\sum_{R\subseteq S\subseteq N} (-1)^{|N\setminus S|} \binom{e[S]}{n/2} = [R = N]\,,$$

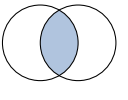
using (1). Thus, F contributes 1 if and only if it covers all nodes. Since F contains $n/2$ edges, F it must be a perfect matching. □

The running time is within a polynomial factor of 2^n, and the space is polynomial; see Fig. 5.

Perspective. Perfect matchings in general graphs is a packing or partitioning problem, while the bipartite case was a a sequencing problem and the graph colouring example in §2 was a covering problem. (Admittedly, the distinction between these things is not very clear.) The application is form [2], which also contains another space–time trade-off based on matrix multiplication.

The point of the large in example in Fig. 5 is to illustrate the intuition that inclusion–exclusion is a *sieve*. We start with a large collection of easy-to-compute objects (the top row in Fig. 5), and let the alternating sum perform a cancellation that sifts through the objects and keeps only the interesting ones in the sieve.

6 Inclusion–exclusion for sets. If two sets A and B have no elements in common, then we have the principle of *addition*: $|A \cup B| = |A| + |B|$. In general, the equality does not hold, and all we have is $|A \cup B| \leq |A| + |B|$. Observing that every element of $A \cap B$ is counted exactly twice on the right hand side allows us subtract the error term:



$$|A \cup B| = |A| + |B| - |A \cap B|\,,$$

often called the *principle of inclusion–exclusion*.

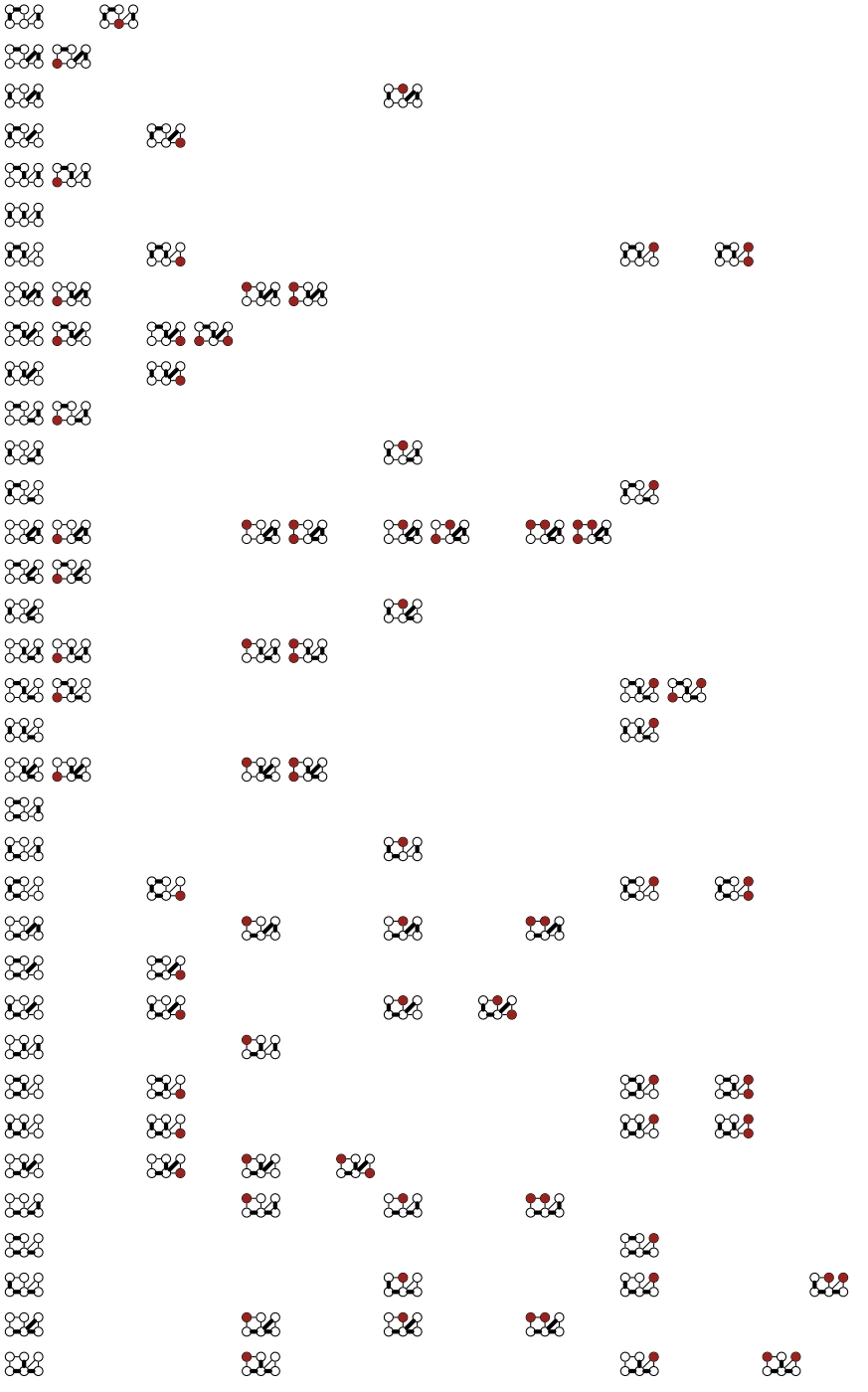
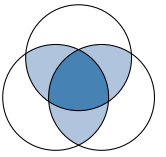


Fig. 5. The perfect matching algorithm for a graph with $n = 6$ and $m = 7$. There are $\binom{7}{3} = 35$ ways to pick 3 edges out of 7, shown in the top row. The triangle appears in 7 other terms (4 negative, 3 positive), the two perfect matchings appear only once.

Actually, that's just a special case, the formula is elevated to a principle by generalising to more sets. For three sets, the formula



$$|A\cup B\cup C| = |A|+|B|+|C|-|A\cap B|-|A\cap C|-|B\cap C|+|A\cap B\cap C|$$

can be verified by staring at a Venn diagram. The right-hand side contains all the possible intersections of A, B, and C, with signs depending on how many sets intersect. Generalising this leads us to

$$|A_1 \cup \cdots \cup A_n| = \sum_{\varnothing\neq S\subseteq N} (-1)^{|S|+1} \Big|\bigcap_{i\in S} A_i\Big|, \tag{8}$$

where $N = \{1,\ldots,n\}$. Equivalently, the number of elements not in any A_i is

$$\big|\overline{A_1 \cup \cdots \cup A_n}\big| = \sum_{S\subseteq N} (-1)^{|S|} \Big|\bigcap_{i\in S} A_i\Big|, \tag{9}$$

with the usual convention that the 'empty' intersection $\bigcap_{i\in\varnothing} A_i$ equals the universe from which the sets are taken.

Proof of (9). We consider the contribution of every element a.

Let $T = \{\, i \in N \colon a \in A_i \,\}$ denote the index set of the sets containing a. The contribution of a to the left hand side of (9) is $[T = \varnothing]$. To determine its contribution to the right hand side, we observe that a belongs to the intersection $\bigcap_{i\in T} A_i$ and all its sub-intersections, so it contributes 1 to all corresponding terms. More precisely, the total contribution of a is given by

$$\sum_{S\subseteq T} (-1)^{|S|} = (-1)^{|T|} \sum_{S\subseteq T} (-1)^{|T\setminus S|} = (-1)^{|T|}[T = \varnothing] = [T = \varnothing]\,,$$

using (11) with $R = \varnothing$. □

Perspective. Expressions (8) and (9) are the standard textbook presentation of inclusion–exclusion. We derived them from (11) with $R = \varnothing$. Let us show the opposite derivation to see that the two formulations are equivalent.

Let T be a nonempty, finite set and write $T = \{1,\ldots,n\}$. Consider the family of identical subsets $A_i = \{1\}$ for all $i \in T$. Their union and every nonempty intersection is $\{1\}$. Thus, from (8),

$$1 = \sum_{\varnothing\neq S\subseteq T} (-1)^{|S|+1} \cdot 1\,,$$

which gives (11) for $R = \varnothing$ after subtracting 1 from both sides.

7 Hamiltonian paths. A *walk* in a graph is a sequence of neighbouring nodes $v_1,\ldots,v_k$. Such a walk is a *path* if every node appears at most once, and a path is *Hamiltonian* if it includes every vertex in G. For ease of notation we also assume that all Hamiltonian paths start in node $v_1 = 1$.

Given a graph G on n nodes N let $a(X)$ denote the number of walks of length n that start in 1 and 'avoid' the nodes in $X \subseteq V$, i.e., walks of the form $1 = v_1, \ldots, v_n$ with $v_i \notin X$ for all $1 \leq i \leq n$. Then the number of Hamiltonian paths in G is $a(\varnothing)$.

Let A_i denote the walks that avoid $\{i\}$. Then $a(\varnothing) = |\bigcup_{i \in N} A_i|$ and $a(X) = |\bigcap_{i \in X} A_i|$. Thus, from (9), we have

$$a(\varnothing) = \sum_{X \subseteq N} (-1)^{|X|} a(X)\,.$$

For every X, the value $a(X)$ can be computed in polynomial time using dynamic programming (over the lengths and endpoints, not over the subsets). For $t \in V$ and $k = 1, \ldots, n$ let for a moment $a^k(X, t)$ denote the number of walks of the form $1 = v_1, \ldots, v_k = t$ with $v_i \notin X$. Then we can set $a^1(X, v) = [v = 1]$ and

$$a^{k+1}(X, t) = \sum_{v \in V} a^k(X, v)[vt \in E]\,.$$

The total time to compute $a(X) = \sum_{t \in V} a(X, t)$ becomes $O(n^2|E|)$, using polynomial space. It follows that Hamiltonicity in an n-node graph can be decided (in fact, counted) in time $O(2^n n^2 |E|)$ and linear space.

Perspective. Hamiltonicity is one of the earliest explicitly algorithmic applications of inclusion–exclusion. It appears in [12], but implicitly already in [14], where it is described for the traveling salesman problem with bounded integer weights. Both these papers have lived in relative obscurity until recently, for example the TSP result has been both reproved and called 'open' in a number of places.

Hamiltonicity is also the canonical application of another algorithmic technique, *dynamic programming over the subsets* [1,11], which yields an algorithm with similar time bounds but exponential space. Thus, we can observe a curious cultural difference in the default approach to hard sequencing problems: Dynamic programming is the well-known solution to Hamiltonicity, while the inclusion–exclusion formulation is often overlooked. For the permanent (§4), the situation is reversed.

8 Steiner tree. For a graph $G = (N, E)$ and a subset $\{t_1, \ldots, t_k\} \subseteq N$ of nodes called *terminals*, a *Steiner tree* is a tree in G that contains all terminals. We want to determine the smallest size of such a tree.

We consider a related structure that is to a tree what a walk is to a path. A *willow* W consists of a multiset of nodes $S(W)$ from N and a parent function $p\colon S(W) \to S(W)$, such that repeated applications of p end in a specified *root* node $r \in S(W)$. The size of W is the number of nodes in $S(W)$, counted with repetition.

Every tree can be turned into a willow, by selecting an arbitrary root and orienting the edges towards it, but the converse is not true. However, a minimum

size willow over a set of nodes is a tree: Assume a node appears twice in W. Remove one of its occurrences $u \in S(W)$, not the root node, and change the parent $p(v)$ of all v with $p(v) = u$ to $p(v) = p(u)$. The resulting willow is smaller than W but spans the same nodes. Finally, when all repeated nodes are removed, p defines a tree.

Thus, it suffices to look for a size-l willow W that includes all terminals, for increasing $l = k, \ldots, n$. Set A_i to be the set of willows of size l that avoid t_i. Then, from (9), the number of willows of size l that include all terminals is

$$\sum_{X \subseteq K} (-1)^{|X|} a^l(X)\,,$$

where $a^l(X) = |\bigcap_{i \in X} A_i|$ is the number of willows of size l that avoid the terminals in X.

Again, we can use dynamic programming to compute $a^l(X)$ for given X. For all $X \subseteq V$ and $u \notin X$ let $a^l(X, u)$ denote the number willows of size l that avoid X and whose root is $u \notin X$. Then $a^1(X, u) = 1$ and

$$a^k(X, u) = \sum_{uv \in E} \sum_{i=1}^{k-1} a^i(X, u) a^{k-i}(X, v)\,.$$

Perspective. This application is from [16]. The role of inclusion–exclusion is slightly different from the Hamiltonian path construction in the previous subsection, because we have no control over the size of the objects we are sieving for.

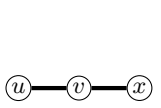
There, we sifted through all walks of length n. What was left in the sieve were the walks of length n that visit all n nodes. Thus, every node appears exactly once, so that sieve contained exactly the desired solutions, i.e., the Hamiltonian paths.

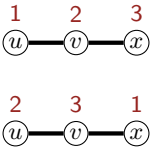
Here, we sift through all willows of size l. What is left in the sieve are the willows that visit all terminals. For given l, these are not necessarily trees. Instead, correctness hinges on the fact that we already sifted through willows of smaller size. To strain the metaphor, we use increasingly fine sieves until we find something.

9 Long paths. Consider a graph $G = (N, E)$ and integer $k \leq n$. We want to detect if G has a path of length k. Inspired by the Hamiltonicity construction in §7 we look at all walks $(w_1, \ldots, w_k)$ on k nodes. For expository reasons we again stipulate that all walks begin in a fixed node $w_1 = 1$. Write $K = \{1, \ldots, k\}$.

For every edge e pick a random value $r(e)$. For every vertex v and integer $k \in K$ pick a random value $r(v, k)$. With foresight, the values are chosen uniformly at random form a finite field F of characteristic 2 and size at least $2k(k-1)$. All computation is in this field. For every walk $W = (w_1, \ldots, w_k)$ starting in $w_1 = 1$ and every function $\phi\colon K \to K$ define the term

$$p(W, \phi) = \left(\prod_{i=1}^{k-1} r(w_i w_{i+1}) \right) \left(\prod_{i=1}^{k} r(w_i, \phi(i)) \right), \tag{10}$$





$$r(uv) \cdot r(vx) \cdot r(u,1) \cdot r(v,2) \cdot r(x,3)$$

$$r(uv) \cdot r(vx) \cdot r(u,2) \cdot r(v,3) \cdot r(x,1)$$

Fig. 6. Left: The path $W = (u, v, x)$. Middle: The nodes of W labelled with two permutations. Right: The terms associated with W and the two permutations.

see Fig. 6.

Consider the sum over all walks W in G and all permutations $\pi \in S_k$,

$$p(G) = \sum_{\pi \in S_k} \sum_{W} p(W, \pi)\,. \tag{11}$$

We will show below that

$$\Pr(\, p(G) = 0\,) \begin{cases} < \frac{1}{2}\,, & \text{if } G \text{ contains a } k\text{-path}\,; \\ = 0\,, & \text{otherwise}\,. \end{cases} \tag{12}$$

where the probability is taken over the random choices of r.

To compute (11), we first recognise a summation over a permutation and replace it by an alternating sum over functions with restricted range, as in (6):

$$\sum_{\pi \in S_k} \sum_{W} p(W, \pi) = \sum_{S \subseteq K} (-1)^{|K \setminus S|} \sum_{\phi\colon K \to S} \sum_{W} p(W, \phi)\,.$$

For each $S \subseteq K$, the value of the two inner sums can be computed efficiently using dynamic programming; we omit the details. The total running time is within a polynomial (in n) factor of 2^k.

Proof of (12). Consider the contribution of every walk W to (11).

First assume that W is non-simple and let π be a permutation. We will construct anther permutation ρ such that $\pi \neq \rho$ but $p(W, \pi) = -p(W, \rho)$. Thus, summing over all permutations, the contribution of W is even and therefore vanishes in F. To construct ρ, let (i, j) be the first self-intersection on W, i.e., the lexicographically minimal pair with $w_i = w_j$ and $i < j$. Set ρ equal to π except for $\rho(i) = \pi(j)$ and $\rho(j) = \pi(i)$.

Now assume that W is a path. It is useful to view (11) as a polynomial in variables $x(e), x(v, k)$, evaluated at random points $x(e) = r(e)$, $x(v, k) = r(v, k)$. For every permutation π, the monomial

$$\left(\prod_{i=1}^{k-1} x(w_i w_{i+1})\right)\left(\prod_{i=1}^{k-1} x(w_i, \pi(i))\right)$$

is unique. To see this, both W and π can be recovered from $p(W,\pi)$ by first reconstructing the nodes $w_1,\dots,w_k$ in order, starting at $w_1 = 1$ and following the unique incident edge described by the terms $x(e)$, and then reconstructing π from the terms $x(w_i,\pi(i))$. Thus, (11) can be viewed as a nonzero polynomial of degree $k(k-1)$ evaluated at $m+nk$ random points from F. By the DeMillo–Lipton–Schwarz–Zippel lemma [9,18], it evaluates to zero with probability less than $k(k-1)/|F| \le \frac{1}{2}$. □

Perspective. The construction is implicit in [6] and not optimal. Again, the starting point is the same as for Hamiltonicity in §7: to sieve for paths among walks, whose contribution is computed by dynamic programming.

However, instead of counting the number of walks, we define an associated family of algebraic objects (namely, a multinomial defined by the walk and a permutation) and work with these objects instead. Strictly speaking, we did associate algebraic objects to walks even before, but the object was somewhat innocent: the integer 1.

There are two filtering processes at work: The sifting for paths among walks is performed by the the cancellation of non-simple, permutation-labelled walks in characteristic 2, rather than the by inclusion–exclusion sieve. At the danger of overtaxing the sieving metaphor, the permutation-labelling plays the role of *mercury* in gold mining; inclusion–exclusion ensures that the 'mercury' can be added and removed in time 2^k instead of the straightforward $k!$.

10 Yates's algorithm. Let $f\colon 2^N \to \{0,1\}$ be the the indicator function of the nonempty independent sets in a graph. We will revisit the task of §3, computing

$$g(S) = \sum_{R\subseteq S} f(R)\,, \tag{13}$$

for all $S \subseteq N$.

The computation proceeds in rounds $i = 1,\dots,n$. Initially, set $g_0(S) = f(S)$ for all $S \subseteq N$. Then we have, for $i = 1,\dots,n$,

$$g_i(S) = g_{i-1}(S) + [i \in S]\cdot g_{i-1}(S\setminus\{i\}) \qquad (S\subseteq N)\,. \tag{14}$$

Finally, $g(S) = g_n(S)$.

Proof of (14). The intuition is that $g_0(S),\dots,g_n(S)$ approach $g(S)$ 'coordinate-wise' by fixing fewer and fewer bits of S. To be precise, for $i = 1,\dots,n$,

$$g_i(S) = \sum_{R\subseteq S} [S\cap\{i+1,\dots,n\} = R\cap\{i+1,\dots,n\}]\cdot f(R)\,. \tag{15}$$

In particular, $g_n(S) = \sum_{R\subseteq S} f(R)$. Correctness of (14) is established by a straightforward but tedious induction argument for (15). The base case $g_0 = f$ is

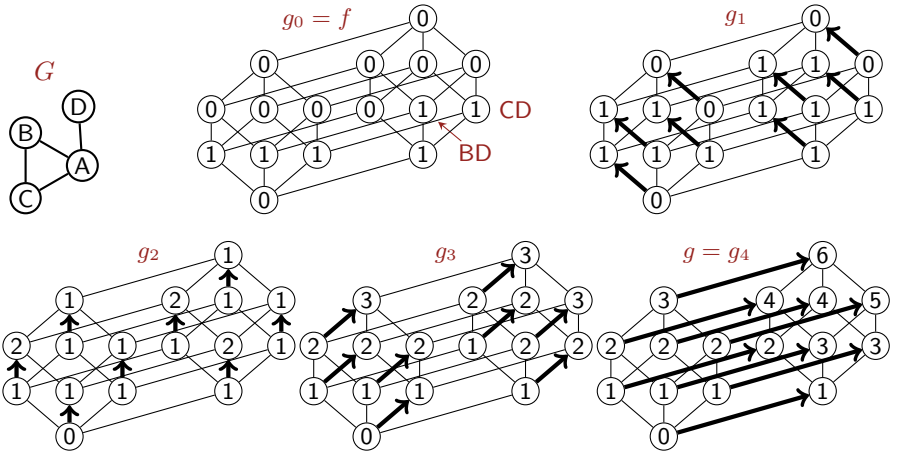


Fig. 7. Yates's algorithm on the indicator function of the nonempty independent sets of the graph G. Arrows indicate how the value of $g_i(S)$ for $i \in S$ is computed by adding $g_{i-1}(S \setminus \{i\})$ to the 'previous' value $g_{i-1}(S)$.

immediate. For the inductive step, adopt the notation $S(i)$ for $S \cap \{i+1, \ldots, n\}$. Then the right hand side of (15) can be written as

$$\begin{aligned} &\sum_{R \subseteq S} [S(i) = R(i)] f(R) \\ &= \sum_{\substack{R \subseteq S \\ i \in R}} [S(i) = R(i)] f(R) + \sum_{\substack{R \subseteq S \\ i \notin R}} [S(i) = R(i)] f(R) \,. \end{aligned}$$

If $i \notin S$ then the first sum vanishes and the second sum simplifies to

$$\sum_{R \subseteq S} [S(i-1) = R(i-1)] f(R) = g_{i-1}(S)$$

by induction. If $i \in S$ then we can rewrite both sums to

$$\begin{aligned} &\sum_{R \subseteq S} [S(i-1) = R(i-1)] f(R) + \sum_{\substack{R \subseteq S \\ i \notin S}} [S(i-1) = R(i-1)] f(R) \\ &\qquad = g_{i-1}(S) + g_{i-1}(S \setminus \{i\}) \end{aligned}$$

by induction. Finally, by (14) the entire expression equals $g_i(S)$. □

Perspective. As before, our approach is basically dynamic programming for a decrease-and-conquer recurrence. The time and space requirements are within a linear factor of the ones given in §2. However, the expression (14) is more general and does not depend on the structure of independent sets. It applies

to any function $f\colon 2^N \to R$ from subsets to a ring, extending the algorithm to many other covering problems than graph colouring.

Yates's algorithm has much in common with the fast Fourier transform. We can illustrate its operation using a butterfly-like network, see Fig. 8.

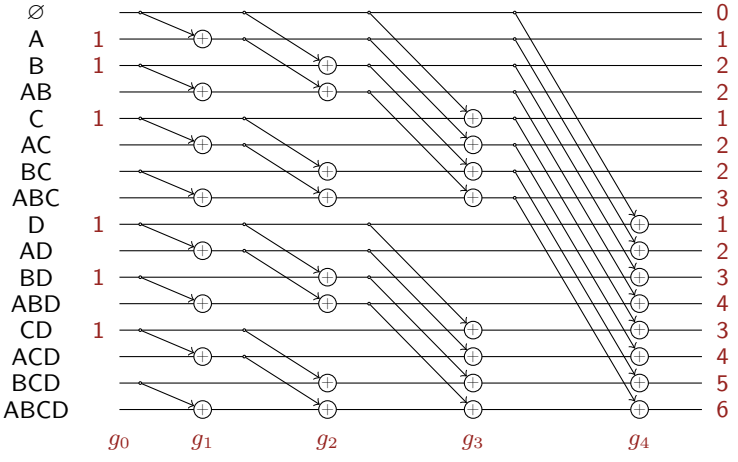


Fig. 8. Yates's algorithm for the zeta transform

Here we used Yates's algorithm to compute (13), but the method is more general than that. For example, it computes the *Möbius transform*, see (17) below, and many others. A classical treatment of the algorithm appears in [13], recent applications and modifications are in [7] and the forthcoming journal version of [3].

11 Möbius inversion. Let $f\colon 2^N \to \{0,1\}$ be a function of subsets of N to $\{0,1\}$ (indeed, any ring would do). To connect to the graph colouring example from §2, think of f as the indicator function of the nonempty independent sets in a graph. The *zeta transform* of f is the function $(f\zeta)\colon 2^N \to \{0,1\}$ defined point-wise by

$$(f\zeta)(T) = \sum_{S\subseteq T} f(S)\,. \tag{16}$$

The brackets around $(f\zeta)$ are usually omitted. The *Möbius transform* of f is the function $(f\mu)\colon 2^N \to \{0,1\}$ defined point-wise by

$$(f\mu)(T) = \sum_{S\subseteq T} (-1)^{|T\setminus S|} f(S), \tag{17}$$

This allows us to state the principle of inclusion–exclusion in yet another way:

$$f\zeta\mu = f\mu\zeta = f\,. \tag{18}$$

Proof. We show $f\zeta\mu = f$, the other argument is similar.

$$\begin{aligned} f\zeta\mu(T) &= \sum_{S\subseteq T} (-1)^{|T\setminus S|} \sum_{R\subseteq S} f(R) \\ &= \sum_{S}\sum_{R} [S\subseteq T][R\subseteq S](-1)^{|T\setminus S|} f(R) \\ &= \sum_{R} f(R) \sum_{S} [R\subseteq S\subseteq T](-1)^{|T\setminus S|}\,. \end{aligned}$$

By (11), the inner sum equals $[R = T]$, so the expression simplifies to $f(T)$. □

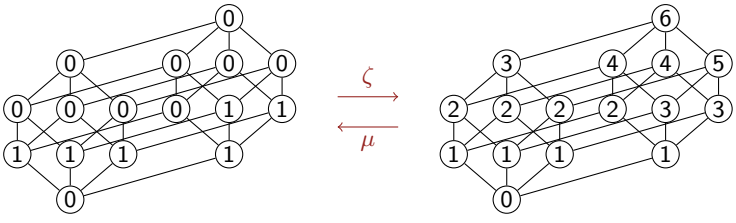


Fig. 9. Möbius inversion

Perspective. For completeness, let us also derive (11) from (18), to see that the two claims are equivalent. Consider two sets R and T. Define $f(Q) = [Q = R]$. Then, expanding (18),

$$\begin{aligned} [R = T] = f(T) &= \sum_{S\subseteq T} (-1)^{|T\setminus S|} \sum_{Q\subseteq S} f(Q) = \sum_{S\subseteq T} (-1)^{|T\setminus S|} [R\subseteq S] \\ &= \sum_{R\subseteq S\subseteq T} (-1)^{|T\setminus S|}\,. \end{aligned}$$

12 Covering by Möbius inversion. We now give alternative argument for the graph colouring expression (2).

Let $f\colon 2^N \to \{0,1\}$ be the indicator function of the nonempty independent sets of a graph $G = (N, E)$. We want to count the number of ways so cover N with k nonempty independent sets. Define $g(S)$ to be the number of ways to choose k nonempty independent sets whose union is S. Then we claim

$$g\zeta = (f\zeta)^k\,.$$

To see this, for every $T \subseteq V$, view $g\zeta(T)$ and $\big(f\zeta(T)\big)^k$ as two different ways of counting the number of ways so select k nonempty independent subsets of T. Now, by Möbius inversion (18), we have

$$g = (f\zeta)^k\mu\,,$$

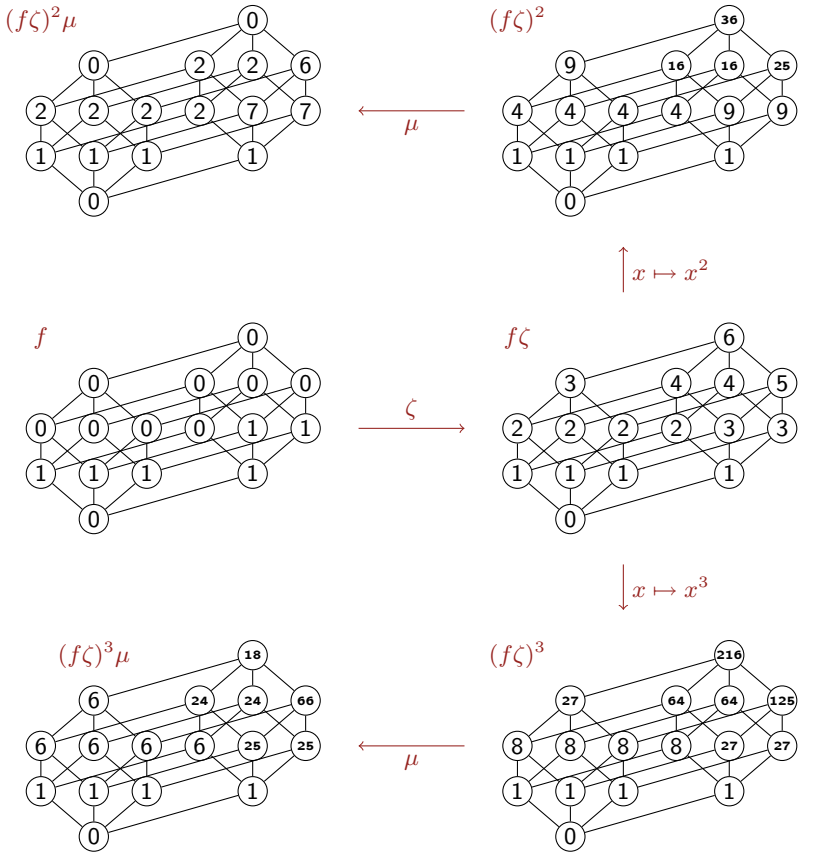


Fig. 10. Covering by Möbius inversion for $k = 2$ and $k = 3$

which is the left hand side of (2). In fact, we can now rewrite and understand the left hand side of (2) as

$$\underbrace{\sum_{S \subseteq N} (-1)^{|N \setminus S|}}_{\mu} \left(\underbrace{\sum_{R \subseteq S} f(R)}_{\zeta} \right)^{k}$$

operation in the transformed domain

function in the original domain

Perspective. The fact that f was the indicator function of the *independent sets* played no role in this argument. It works for a many covering problems, and with some work also for packing and partitioning problems.

Taxonomically, we can view inclusion–exclusion as a *transform-and-conquer* technique, like the Fourier transform. This can be expressed succinctly in terms of Möbius inversion, illustrated in Fig. 10. The zeta transform translates the

original problem into a different domain, where the computation is often easier, and the Möbius transform translates back. In the covering example, the operation in the transformed domain, exponentiation, is particularly simple. The idea extends to many other operations in the transformed domain, see [3].

Concluding remarks. A comprehensive presentation of many of the ideas mentioned here appears in a recent monograph [10], with many additional examples. In particular a whole chapter is devoted to subset convolution, the most glaring omission from the present introduction.

I owe much of my understanding of this topic to illuminating conversations with my collaborators, Andreas Björklund, Petteri Kaski, and Mikko Koivisto.

References

1. Bellman, R.: Dynamic programming treatment of the travelling salesman problem. J. Assoc. Comput. Mach. 9(1), 61–63 (1962)
2. Björklund, A., Husfeldt, T.: Exact algorithms for exact satisfiability and number of perfect matchings. Algorithmica 52(2), 226–249 (2008)
3. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Fourier meets Möbius: fast subset convolution. In: Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, CA, June 11-13, pp. 67–74. ACM, New York (2007)
4. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Covering and packing in linear space. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 727–737. Springer, Heidelberg (2010)
5. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Evaluation of permanents in rings and semirings. Inf. Process. Lett. 110(20), 867–870 (2010)
6. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Narrow sieves for parameterized paths and packings. arXiv:1007.1161 (2010)
7. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Trimmed Moebius inversion and graphs of bounded degree. Theory Comput. Syst. 47(3), 637–654 (2010)
8. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion–exclusion. SIAM J. Comput. 39(2), 546–563 (2009)
9. DeMillo, R.A., Lipton, R.J.: A probabilistic remark on algebraic program testing. Inform. Process Lett. 7, 193–195 (1978)
10. Fomin, F., Kratsch, D.: Exact Exponential Algorithms. Texts in Theoretical Computer Science. Springer, Heidelberg (2010)
11. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. J. Soc. Indust. Appl. Math. 10, 196–210 (1962)
12. Karp, R.M.: Dynamic programming meets the principle of inclusion and exclusion. Oper. Res. Lett. 1(2), 49–51 (1981-1982)
13. Knuth, D.E.: The Art of Computer Programming, 3rd edn. Seminumerical Algorithms, vol. 2. Addison-Wesley, Upper Saddle River (1998)
14. Kohn, S., Gottlieb, A., Kohn, M.: A generating function approach to the traveling salesman problem. In: Proceedings of the 1977 ACM Annual Conference, Seattle, WA, October 17-19, pp. 294–300. ACM, New York (1977)

15. Lawler, E.L.: A note on the complexity of the chromatic number problem. Inf. Process. Lett. 5(3), 66–67 (1976)
16. Nederlof, J.: Fast polynomial-space algorithms using Möbius inversion: Improving on Steiner tree and related problems. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 713–725. Springer, Heidelberg (2009)
17. Ryser, H.J.: Combinatorial Mathematics. The Carus Mathematical Monographs, vol. 14. The Mathematical Association of America, Washington, D.C. (1963)
18. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. J. Assoc. Comput. Mach. 27, 701–717 (1980)

On the Relation between Differential Privacy and Quantitative Information Flow*

Mário S. Alvim, Miguel E. Andrés,
Konstantinos Chatzikokolakis, and Catuscia Palamidessi

INRIA and LIX, Ecole Polytechnique, France

Abstract. Differential privacy is a notion that has emerged in the community of statistical databases, as a response to the problem of protecting the privacy of the database's participants when performing statistical queries. The idea is that a randomized query satisfies differential privacy if the likelihood of obtaining a certain answer for a database x is not too different from the likelihood of obtaining the same answer on adjacent databases, i.e. databases which differ from x for only one individual.

Information flow is an area of Security concerned with the problem of controlling the leakage of confidential information in programs and protocols. Nowadays, one of the most established approaches to quantify and to reason about leakage is based on the Rényi min entropy version of information theory.

In this paper, we analyze critically the notion of differential privacy in light of the conceptual framework provided by the Rényi min information theory. We show that there is a close relation between differential privacy and leakage, due to the graph symmetries induced by the adjacency relation. Furthermore, we consider the utility of the randomized answer, which measures its expected degree of accuracy. We focus on certain kinds of utility functions called "binary", which have a close correspondence with the Rényi min mutual information. Again, it turns out that there can be a tight correspondence between differential privacy and utility, depending on the symmetries induced by the adjacency relation and by the query. Depending on these symmetries we can also build an optimal-utility randomization mechanism while preserving the required level of differential privacy. Our main contribution is a study of the kind of structures that can be induced by the adjacency relation and the query, and how to use them to derive bounds on the leakage and achieve the optimal utility.

1 Introduction

Databases are commonly used for obtaining statistical information about their participants. Simple examples of statistical queries are, for instance, the predominant disease of a certain population, or the average salary. The fact that

* This work has been partially supported by the project ANR-09-BLAN-0169-01 PANDA and by the INRIA DRI Equipe Associée PRINTEMPS. The work of Miguel E. Andrés has been supported by the LIX-Qualcomm postdoc fellowship 2010.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 60–76, 2011.

the answer is publicly available, however, constitutes a threat for the privacy of the individuals.

In order to illustrate the problem, consider a set of individuals *Ind* whose attribute of interest[1] has values in *Val*. A particular database is formed by a subset of *Ind*, where a certain value in *Val* is associated to each participant. A query is a function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ is the set of all possible databases, and $\mathcal{Y}$ is the domain of the answers.

For example, let *Val* be the set of possible salaries and let f represent the query "what is the average salary of the participants in the database". In principle we would like to consider the *global information* relative to a database x as *public*, and the *individual information* about a participant i as *private*. Namely, we would like to be able to obtain $f(x)$ without being able to infer the salary of i. However, this is not always possible. In particular, if the number of participants in x is known (say n), then the removal of i from the database would allow to infer i's salary by querying again the new database x', and by applying the formula $n\, f(x) - (n-1)\, f(x')$. Using an analogous reasoning we can argue that not only the removal, but also the addition of an individual is a threat for his privacy.

Another kind of private information we may want to protect is whether an individual i is participating or not in a database. In this case, if we know for instance that i earns, say $5K$ Euros/month, and all the other individuals in *Ind* earn less than $4K$ Euros/month, then knowing that $f(x) > 5K$ Euros/month will reveal immediately that i is in the database x.

A common solution to the above problems is to introduce some output perturbation mechanism based on *randomization*: instead of the exact answer $f(x)$ we report a "noisy" answer. Namely, we use some randomized function $\mathcal{K}$ which produces values in some domain[2] $\mathcal{Z}$ according to some probability distribution that depends on the input $x \in \mathcal{X}$. Of course for certain distributions it may still be possible to guess the value of an individual with a high probability of success. The notion of *differential privacy*, due to Dwork [10,13,11,12], is a proposal to control the risk of violating privacy for both kinds of threats described above (value and participation). The idea is to say that $\mathcal{K}$ satisfies ϵ-differential privacy (for some $\epsilon > 0$) if the ratio between the probabilities that two adjacent databases give the same answer is bound by e^{ϵ}, where by "adjacent" we mean that the databases differ for only one individual (either for the value of an individual or for the presence/absence of an individual). Often we will abbreviate "ϵ-differential privacy" as ϵ-d.p.

Obviously, the smaller is ϵ, the greater is the privacy protection. In particular, when ϵ is close to 0 the output of $\mathcal{K}$ is nearly independent from the input (all distributions are almost equal). Unfortunately, such $\mathcal{K}$ is practically useless. The *utility*, i.e. the capability to retrieve accurate answers from the reported

[1] In general we could be interested in several attributes simultaneously, and in this case *Val* would be a set of tuples.

[2] The new domain $\mathcal{Z}$ may coincide with $\mathcal{Y}$, but not necessarily. It depends on how the randomization mechanism is defined.

ones, is the other important characteristic of $\mathcal{K}$, and it is clear that there is a trade-off between utility and privacy. On the other hand, these two notions are not the complete opposite of each other, because utility concerns the relation between the reported answer and the real answer, while privacy is concerns the relation between the reported answer and the information in the database. This asymmetry makes more interesting the problem of finding a good compromise between the two.

At this point, we would like to remark an intriguing analogy between the area of differential privacy and that of *quantitative information flow* (QIF), both in the motivations and in the basic conceptual framework. Information flow is concerned with the leakage of secret information through computer systems, and the attribute "quantitative" refers to the fact that we are interested in measuring the amount of leakage, not just its occurrence. One of the most established approaches to QIF is based on information theory: the idea is that a system is seen as a channel in the information-theoretic sense, where the secret is the input and the observables are the output. The entropy of the input represents its vulnerability, i.e. how easy it is for an attacher to guess the secret. We distinguish between the *a priori* entropy (before the observable) and the *a posteriori* entropy (given the observable). The difference between the two gives the *mutual information* and represents, intuitively, the increase in vulnerability due to the observables produced by the system, so it is naturally considered as a measure of the leakage. The notion of entropy is related to the kind of attack we want to model, and in this paper we focus on the Rényi min entropy [18], which represents the so-called *one-try attacks*. In recent years there has been a lot of research aimed at establishing the foundations of this framework [19,7,16,3,5]. It is worth pointing out that the a posteriori Rényi min entropy corresponds to the concept of Bayes risk, which has also been proposed as a measure of the effectiveness of attacks [8,6,17].

The analogy hinted above between differential privacy and QIF is based on the following observations: at the motivational level, the concern about privacy is akin the concern about information leakage. At the conceptual level, the randomized function $\mathcal{K}$ can be seen as an information-theoretic channel, and the limit case of $\epsilon = 0$, for which the privacy protection is total, corresponds to a 0-capacity channel[3] (the rows of the channel matrix are all identical), which does not allow any leakage. Another promising similarity is that the notion of utility (in the binary case) corresponds closely to the Bayes risk.

In this paper we investigate the notion of differential privacy, and its implications, in light of the min-entropy information theoretic framework developed for QIF. In particular, we wish to explore the following natural questions:

1. Does ϵ-d.p. induce a bound on the information leakage of $\mathcal{K}$?
2. Does ϵ-d.p. induce a bound on the information leakage *relative to an individual*?
3. Does ϵ-d.p. induce a bound on the utility?

[3] The channel capacity is the maximum mutual information over all possible input distributions.

4. Given f and ϵ, can we construct a $\mathcal{K}$ which satisfies ϵ-d.p. and maximum utility?

We will see that the answers to (1) and (2) are positive, and we provide bounds that are tight, in the sense that for every ϵ there is a $\mathcal{K}$ whose leakage reaches the bound. For (3) we are able to give a tight bound in some cases which depend on the structure of the query, and for the same cases, we are able to construct an oblivious[4] $\mathcal{K}$ with maximum utility, as requested by (4).

Part of the above results have already appeared in [1], and are based on techniques which exploit the graph structure that the adjacency relation induces on the domain of all databases $\mathcal{X}$, and on the domain of the correct answers $\mathcal{Y}$. The main contribution of this paper is an extension of those techniques, and a coherent graph-theoretic framework for reasoning about the symmetries of those domains. More specifically:

- We explore the graph-theoretic foundations of the adjacency relation, and point out various types of symmetries which allow us to establish a strict link between differential privacy and information leakage.
- We give a tight bound for the question (2) above, strictly smaller than the one in [1].
- We extend the structures for which we give a positive answer to the questions (3) and (4) above. In [1] the only case considered was the class of graphs with single-orbit automorphisms. Here we show that the results hold also for regular-distance graphs and a variant of vertex-transtive graphs.

In this paper we focus on the case in which $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ are finite, leaving the more general case for future work.

2 Preliminaries

2.1 Database Domain and Differential Privacy

Let *Ind* be a finite set of individuals that may participate in a database and *Val* a finite set of possible values for the attribute of interest of these individuals. In order to capture in a uniform way the presence/absence of an individual in the database, as well as its value, we enrich the set of possible values with an element a representing the absence of the individual. Thus the set of all possible databases is the set $\mathcal{X} = V^{Ind}$, where $V = Val \cup \{a\}$. We will use u and v to denote the cardinalities of *Ind* and V, $|Ind|$ and $|V|$, respectively. Hence we have that $|\mathcal{X}| = v^u$. A database x can be represented as a u-tuple $v_0 v_1 \dots v_{u-1}$ where each $v_i \in V$ is the value of the corresponding individual. Two databases x, x' are *adjacent* (or *neighbors*), written $x \sim x'$, if they differ for the value of exactly one individual. For instance, for $u = 3$, $v_0 v_1 v_2$ and $v_0 w_1 v_2$, with $w_1 \neq v_1$, are adjacent. The structure $(\mathcal{X}, \sim)$ forms an undirected graph.

[4] A randomized function $\mathcal{K}$ is oblivious if its probability distribution depends only on the answer to the query, and not on the database.

Intuitively, differential privacy is based on the idea that a randomized query function provides sufficient protection if the ratio between the probabilities of two adjacent databases to give a certain answer is bound by e^ϵ, for some given $\epsilon > 0$. Formally:

Definition 1 ([12]). *A randomized function $\mathcal{K}$ from $\mathcal{X}$ to $\mathcal{Z}$ satisfies ϵ-differential privacy if for all pairs $x, x' \in \mathcal{X}$, with $x \sim x'$, and all $S \subseteq \mathcal{Z}$, we have that:*

$$Pr[\mathcal{K}(x) \in S] \leq e^\epsilon \times Pr[\mathcal{K}(x') \in S]$$

The above definition takes into account the possibility that $\mathcal{Z}$ is a continuous domain. In our case, since $\mathcal{Z}$ is finite, the probability distribution is discrete, and we can rewrite the property of ϵ-d.p. more simply as (using the notation of conditional probabilities, and considering both quotients):

$$\frac{1}{e^\epsilon} \leq \frac{Pr[Z = z|X = x]}{Pr[Z = z|X = x']} \leq e^\epsilon \qquad \text{for all } x, x' \in \mathcal{X} \text{ with } x \sim x', \text{ and all } z \in \mathcal{Z}$$

where X and Z represent the random variables associated to $\mathcal{X}$ and $\mathcal{Z}$, respectively.

2.2 Information Theory and Application to Information Flow

In the following, X, Y denote two discrete random variables with carriers $\mathcal{X} = \{x_0, \ldots, x_{n-1}\}$, $\mathcal{Y} = \{y_0, \ldots, y_{m-1}\}$, and probability distributions $p_X(\cdot)$, $p_Y(\cdot)$, respectively. An information-theoretic channel is constituted by an input X, an output Y, and the matrix of conditional probabilities $p_{Y|X}(\cdot \mid \cdot)$, where $p_{Y|X}(y \mid x)$ represent the probability that Y is y given that X is x. We shall omit the subscripts on the probabilities when they are clear from the context.

Rényi min-entropy. In [18], Rényi introduced an one-parameter family of entropy measures, intended as a generalization of Shannon entropy. The Rényi entropy of order α ($\alpha > 0$, $\alpha \neq 1$) of a random variable X is defined as $H_\alpha(X) = \frac{1}{1-\alpha} \log_2 \sum_{x \in \mathcal{X}} p(x)^\alpha$. We are particularly interested in the limit of H_α as α approaches ∞. This is called *min-entropy*. It can be proven that $H_\infty(X) \stackrel{\text{def}}{=} \lim_{\alpha \to \infty} H_\alpha(X) = -\log_2 \max_{x \in \mathcal{X}} p(x)$.

Rényi defined also the α-generalization of other information-theoretic notions, like the Kullback-Leibler divergence. However, he did not define the α-generalization of the conditional entropy, and there is no general agreement on what it should be. For the case $\alpha = \infty$, we adopt here the definition of conditional entropy proposed by Smith in [19]:

$$H_\infty(X \mid Y) = -\log_2 \sum_{y \in \mathcal{Y}} p(y) \max_{x \in \mathcal{X}} p(x \mid y) \tag{1}$$

Analogously to the Shannon case, we can define the Rényi-mutual information I_∞ as $H_\infty(X) - H_\infty(X \mid Y)$, and the capacity C_∞ as $\max_{p_X(\cdot)} I_\infty(X; Y)$. It has

been proven in [7] that C_∞ is obtained at the uniform distribution, and that it is equal to the sum of the maxima of each column in the channel matrix, i.e., $C_\infty = \sum_{y \in \mathcal{Y}} \max_{x \in \mathcal{X}} p(y \mid x)$.

Interpretation in terms of attacks: Rényi min-entropy can be related to a model of adversary who is allowed to ask exactly one question, which must be of the form "is $X = x$?" (one-try attacks). More precisely, $H_\infty(X)$ represents the (logarithm of the inverse of the) probability of success for this kind of attacks and with the best strategy, which consists, of course, in choosing the x with the maximum probability.

As for $H_\infty(X \mid Y)$, it represents the inverse of the (expected value of the) probability that the same kind of adversary succeeds in guessing the value of X *a posteriori*, i.e. after observing the result of Y. The complement of this probability is also known as *Bayes risk*. Since in general X and Y are correlated, observing Y increases the probability of success. Indeed we can prove formally that $H_\infty(X \mid Y) \leq H_\infty(X)$, with equality if and only if X and Y are independent. $I_\infty(X;Y)$ corresponds to the *ratio* between the probabilities of success a priori and a posteriori, which is a natural notion of leakage. Note that $I_\infty(X;Y) \geq 0$, which seems desirable for a good notion of leakage.

3 Graph Symmetries

In this section we explore some classes of graphs that allow us to derive a strict correspondence between ϵ-d.p. and the a posteriori entropy of the input.

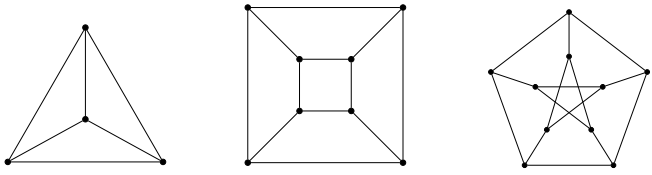
Let us first recall some basic notions. Given a graph $G = (\mathcal{V}, \sim)$, the *distance* $d(v, w)$ between two vertices $v, w \in \mathcal{V}$ is the number of edges in a shortest path connecting them. The *diameter* of G is the maximum distance between any two vertices in $\mathcal{V}$. The degree of a vertex is the number of edges incident to it. G is called *regular* if every vertex has the same degree. A regular graph with vertices of degree k is called a k-regular graph. An authomorphism of G is a permutation σ of the vertex set $\mathcal{X}$, such that for any pair of vertices x, x', if $x \sim x'$, then $\sigma(x) \sim \sigma(x')$. If σ is an authomorphism, and v a vertex, the orbit of v under σ is the set $\{v, \sigma(v), \ldots, \sigma^{k-1}(v)\}$ where k is the smallest positive integer such that $\sigma^k(v) = v$. Clearly, the orbits of the vertices under σ define a partition of $\mathcal{V}$.

The following two definition introduce the classes of graphs that we are interested in. The first class is well known in literature.

Definition 2. *Given a graph $G = (\mathcal{V}, \sim)$, we say that G is distance-regular if there exist integers $b_i, c_i, i = 0, ..., d$ such that for any two vertices v, w in $\mathcal{V}$ with distance $i = d(v, w)$, there are exactly c_i neighbors of w in $G_{i-1}(x)$ and b_i neighbors of v in $G_{i+1}(x)$, where $G_i(x)$ is the set of vertices y of G with $d(x, y) = i$.*

Some examples of distance-regular graphs are illustrated in Figure 1.

The next class is a variant of the VT (vertex-transitive) class:

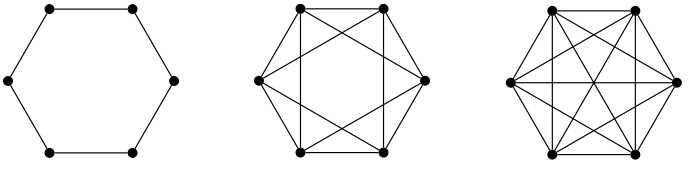


(a) Tetrahedral graph (b) Cubical graph (c) Petersen graph

Fig. 1. Some distance-regular graphs with degree 3

Definition 3. *A graph $G = (\mathcal{V}, \sim)$ is VT^+ (vertex-transitive +) if there are n automorphisms σ_0, σ_1, $\ldots \sigma_{n-1}$, where $n = |\mathcal{V}|$, such that, for every vertex $v \in \mathcal{V}$, we have that $\{\sigma_i(v) \mid 0 \leq i \leq n-1\} = \mathcal{V}$.*

In particular, the graphs for which there exists an authomorphism σ which induces only one orbit are VT^+: in fact it is sufficient to define $\sigma_i = \sigma^i$ for all i from 0 to $n-1$. Figure 2 illustrates some graphs with a single-orbit automorphism.



(a) Cycle: degree 2. (b) Degree 4. (c) Clique: degree 5.

Fig. 2. Some VT^+ graphs

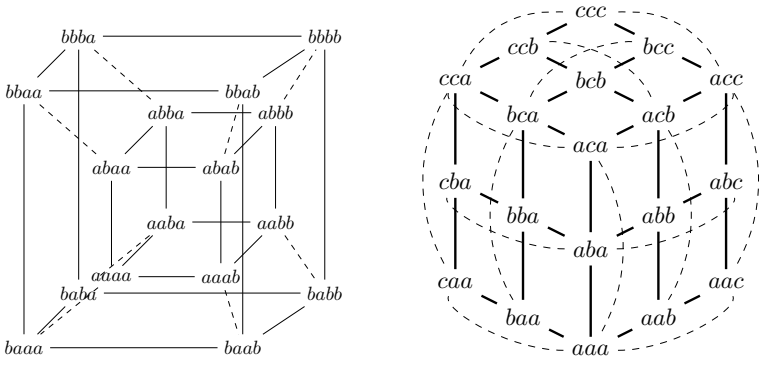
From graph theory we know that neither of the two classes subsumes the other. They have however a non-empty intersection, which contains in particular all the structures of the form $(V^{Ind}, \sim)$, i.e. the database domains.

Proposition 1. *The structure $(\mathcal{X}, \sim) = (V^{Ind}, \sim)$ is both a distance-regular graph and a VT^+ graph.*

Figure 3 illustrates some examples of structures $(V^{Ind}, \sim)$. Note that when $|Ind| = n$ and $|V| = 2$, $(V^{Ind}, \sim)$ is the n-dimentional hypercube.

The situation is summarized in Figure 4. We remark that in general the graphs $(V^{Ind}, \sim)$ do not have a single-orbit authomorphism. The only exceptions are the two simplest structures ($|V| = 2, |Ind| \leq 2$).

The two symmetry classes defined above, distance-regular and VT^+, will be used in the next section to transform a generic channel matrix into a matrix with a symmetric structure, while preserving the a posteriori min entropy and the ϵ-d.p.. This is the core of our technique to establish the relation between differential privacy and quantitive information flow, depending on the structure induced by the database adjacency relation.



(a) $|Ind| = 4, V = \{a, b\}$ (4-dimensional hypercube)

(b) $|Ind| = 3, V = \{a, b, c\}$ (for readability sake we show only part of the graph)

Fig. 3. Some $(V^{Ind}, \sim)$ graphs

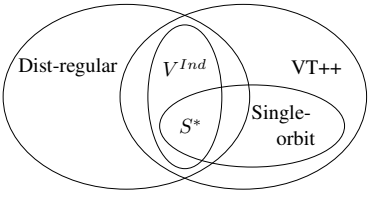


Fig. 4. Venn diagram for the classes of graphs considered in this section. Here, $S^* = \{V^{Ind} \mid |V| = 2, |Ind| \leq 2\}$

4 Deriving the Relation between Differential Privacy and QIF on the Basis of the Graph Structure

This section contains the main technical contribution of the paper: a general technique for determining the relation between ϵ-differential privacy and leakage, and between ϵ-differential privacy and utility, depending on the graph structure induced by $\sim$ and f. The idea is to use the symmetries of the graph structure to transform the channel matrix into an equivalent matrix with certain regularities, which allow to establish the link between ϵ-differential privacy and the a posteriori min entropy.

Let us illustrate briefly this transformation. Consider a channel whose matrix M has at least as many columns as rows. First, we transform M into a matrix M' in which each of the first n columns has a maximum in the diagonal, and the remaining columns are all 0's. Second, under the assumption that the input domain is distance-regular or VT^+, we transform M' into a matrix M'' whose diagonal elements are all the same, and coincide with the maximum element of M'', which we denote here by $\max^{M''}$. These steps are illustrated in Figure 5.

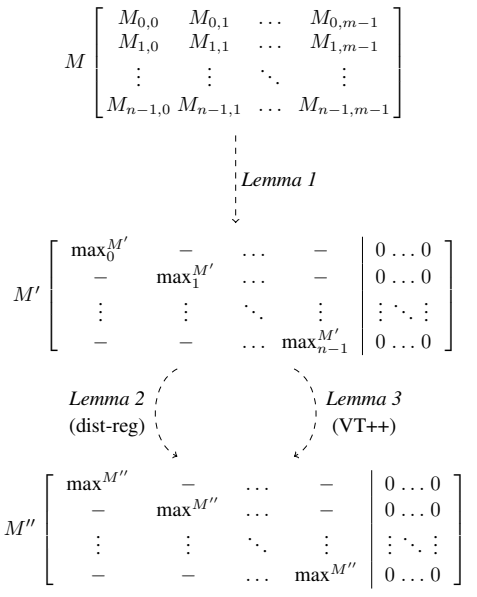


Fig. 5. Matrix transformations for distance-regular and VT^+ graphs

We are now going to present formally our the technique. Let us first fix some notation: In the rest of this section we consider channels with input A and output B, with carriers $\mathcal{A}$ and $\mathcal{B}$ respectively, and we assume that the probability distribution of A is uniform. Furthermore, we assume that $|\mathcal{A}| = n \leq |\mathcal{B}| = m$. We also assume an adjacency relation $\sim$ on $\mathcal{A}$, i.e. that $(\mathcal{A}, \sim)$ is an undirected graph structure. With a slight abuse of notation, we will also write $i \sim h$ when i and h are associated to adjacent elements of $\mathcal{A}$, and we will write $d(i, h)$ to denote the distance between the elements of $\mathcal{A}$ associated to i and h.

We note that a channel matrix M satisfies ϵ-d.p. if for each column j and for each pair of rows i and h such that $i \sim h$ we have that:

$$\frac{1}{e^\epsilon} \leq \frac{M_{i,j}}{M_{h,j}} \leq e^\epsilon.$$

The a posteriori entropy of a channel with matrix M will be denoted by $H_\infty^M(A|B)$.

Next Lemma is relative to the first step of the transformation.

Lemma 1. *Consider a channel with matrix M. Assume that M satisfies ϵ-d.p.. Then it is possible to transform M into a matrix M' such that:*

- *Each of the first n columns has a maximum in the diagonal, i.e. $M'_{i,i} = \max_i^{M'} = \max_h M'_{h,i}$ for each i from 0 to $n-1$.*
- *The rest of the columns contain only 0's, i.e. $M'_{i,j} = 0$ for each i from 0 to $n-1$ and each j from n to $m-1$.*
- *M' satisfies ϵ-d.p.*
- *$H_\infty^{M'}(A|B) = H_\infty^M(A|B)$.*

Next lemma is relative to the second step of the transformation, for the case of distance-regular graphs.

Lemma 2. *Consider a channel with matrix M'. Assume that M' satisfies ϵ-d.p., and the first n columns have maxima in the diagonal, and the rest of the columns contain only 0's. Assume that $(\mathcal{A}, \sim)$ is distance-regular. Then it is possible to transform M' into a matrix M'' such that:*

- *The elements of the diagonal are all the same, and are equal to the maximum of the matrix, i.e. $M''_{i,i} = \max^{M''} = \max_{h,i} M''_{h,i}$ for each i from 0 to $n-1$.*
- *The rest of the columns contain only 0's.*
- *M'' satisfies ϵ-d.p.*
- *$H_\infty^{M''}(A|B) = H_\infty^{M'}(A|B)$.*

Next lemma is relative to the second step of the transformation, for the case of VT^+ graphs.

Lemma 3. *Consider a channel with matrix M' satisfying the assumptions of Lemma 2, except for the assumption about distance-regularity, which we replace by the assumption that $(\mathcal{A}, \sim)$ is VT^+. Then it is possible to transform M' into a matrix M'' with the same properties as in Lemma 2.*

Note that the fact that in M'' the diagonal elements are all equal to the maximum $\max^{M''}$ implies that $H_\infty^{M''}(A|B) = \max^{M''}$.

Once we have a matrix with the properties of M'', we can use again the graph structure of $\mathcal{A}$ to determine a bound on $H_\infty^{M''}(A|B)$.

First we note that the property of ϵ-d.p. induces a relation between the ratio of elements at any distance:

Remark 1. Let M be a matrix satisfying ϵ-d.p.. Then, for any column j, and any pair of rows i and h we have that:

$$\frac{1}{e^{\epsilon\, d(i,h)}} \leq \frac{M_{i,j}}{M_{h,j}} \leq e^{\epsilon\, d(i,h)}$$

In particular, if we know that the diagonal elements of M are equal to the maximum element $\max^M$, then for each element $M_{i,j}$ we have that:

$$M_{i,j} \geq \frac{\max^M}{e^{\epsilon\, d(i,j)}} \tag{2}$$

Let us fix a row, say row r. For each distance d from 0 to the diameter of the graph, let n_d be the number of elements $M_{r,j}$ that are at distance d from the corresponding diagonal element $M_{j,j}$, i.e. such that $d(r,j) = d$. (Clearly, n_d depends on the structure of the graph.) Since the elements of the row i represent a probability distribution, we obtain the following dis-equation:

$$\max^M \sum_d \frac{n_d}{e^{\epsilon\, d}} \leq 1$$

from which we derive immediately a bound on the min a-posteriori entropy.

Putting together all the steps of this section, we obtain our main result.

Theorem 1. *Consider a matrix M, and let r be a row of M. Assume that $(\mathcal{A}, \sim)$ is either distance-regular or VT$^+$, and that M satisfies ϵ-d.p. For each distance d from 0 to the diameter of $(\mathcal{A}, \sim)$, let n_d be the number of nodes j at distance d from r. Then we have that:*

$$H_\infty^M(A|B) \leq -\log_2 \frac{1}{\sum_d \frac{n_d}{e^{\epsilon\, d}}} \tag{3}$$

Note that this bound is tight, in the sense that we can build a matrix for which (3) holds with equality. It is sufficient to define each element $M_{i,j}$ according to (2) (with equality instead of dis-equality, of course).

In the next section, we will see how to use this theorem for establishing a bound on the leakage and on the utility.

5 Application to Leakage

As already hinted in the introduction, we can regard $\mathcal{K}$ as a channel with input X and output Z. From Proposition 1 we know that $(\mathcal{X}, \sim)$ is both distance-regular and VT$^+$, we can therefore apply Theorem 1. Let us fix a particular database $x \in \mathcal{X}$. The number of databases at distance d from x is

$$n_d = \binom{u}{d} (v-1)^d \tag{4}$$

where $u = |Ind|$ and $v = V$. In fact, recall that x can be represented as a u-tuple with values in V. We need to select d individuals in the u-tuple and then change their values, and each of them can be changed in $v-1$ different ways.

Using the n_d from (4) in Theorem 1 we obtain a binomial expansion in the denominator, namely:

$$H_\infty^M(X|Z) \leq -\log_2 \frac{1}{\sum_{d=0}^{u} \binom{u}{d} (v-1)^d \frac{e^{\epsilon(u-d)}}{e^{\epsilon\, u}}} = -u \log_2 \frac{e^\epsilon}{v-1+e^\epsilon}$$

which gives the following result:

Theorem 2. *If $\mathcal{K}$ satisfies ϵ-d.p., then for the uniform input distribution the information leakage is bound from above as follows:*

$$I_\infty(X;Z) \leq -u \log_2 \frac{v\, e^\epsilon}{v-1+e^\epsilon}$$

We consider now the *leakage for a single individual*. Let us fix a database x, and a particular individual i in Ind. The possible ways in which we can change the value of i in x are $v-1$. All the new databases obtained in this way are adjacent to each other, i.e. the graph structure associated to the input is a clique

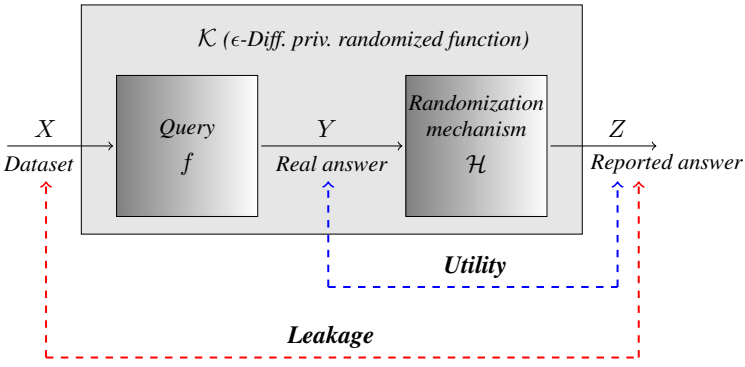


Fig. 6. Schema of an oblivious randomized function

of v nodes. Therefore we obtain $n_d = 1$ for $d = 0$, $n_d = v - 1$ for $d = 1$, and $n_d = 0$ otherwise. By substituting this value of n_d in Theorem 1, we get

$$H_\infty^{ind}(\mathit{Val}|Z) \leq -\log_2 \frac{1}{1 + \frac{v-1}{e^\epsilon}} = -\log_2 \frac{e^\epsilon}{v - 1 + e^\epsilon}$$

which leads to the following result:

Proposition 2. *Assume that $\mathcal{K}$ satisfies ϵ-d.p.. Then for the uniform distribution on V the information leakage for an individual is bound from above as follows:*

$$I_\infty^{ind}(\mathit{Val}; B) \leq \log_2 \frac{v\, e^\epsilon}{v - 1 + e^\epsilon}$$

Note that the bound on the leakage for an individual does not depend on the size of *Ind*, nor on the database x that we fix.

6 Application to Utility

We turn now our attention to the issue of *utility*. We focus on the case in which $\mathcal{K}$ is *oblivious*, which means that it depends only on the (exact) answer to the query, i.e. on the value of $f(x)$, and not on x.

An oblivious function can be decomposed in the concatenation of two channels, one representing the function f, and the other representing the randomization mechanism $\mathcal{H}$ added as output perturbation. The situation is illustrated in Figure 6.

The standard way to define utility is by means of *guess* and *gain* functions. The functionality of the first is $\mathit{guess} : \mathcal{Z} \to \mathcal{Y}$, and it represents the user's strategy to retrieve the correct answer form the reported one. The functionality of the latter is $\mathit{gain} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. the value $\mathit{gain}(y, y')$ represents the reward for guessing the answer y when the correct answer is y'. The utility $\mathcal{U}$ can then be defined as the expected gain:

$$\mathcal{U}(Y, Z) = \sum_{y,z} p(y, z)\, gain(guess(z), y)$$

We focus here on the so-called *binary* gain function, which is defined as

$$gain(y, y') = \begin{cases} 1 & \text{if } y = y' \\ 0 & \text{otherwise} \end{cases}$$

This kind of function represents the case in which there is no reason to prefer an answer over the other, except if it is the *right* answer. More precisely, we get a gain if and only if we guess the right answer.

If the gain function is binary, and the *guess* function represents the user's best strategy, i.e. it is chosen to optimize utility, then there is a well-known correspondence between $\mathcal{U}$ and the Bayes risk / the a posteriori min entropy. Such correspondence is expressed by the following proposition:

Proposition 3. *Assume that gain is binary and guess is optimal. Then:*

$$\mathcal{U}(Y, Z) = \sum_{z} \max_{y}(p(z|y)\, p(y)) = 2^{-H_\infty(Y|Z)}$$

In order to analyze the implications of the ϵ-d.p. requirement on the utility, we need to consider the structure that the adjacency relation induces on $\mathcal{Y}$. Let us define $\sim$ on $\mathcal{Y}$ as follows: $y \sim y'$ if there are $x, x' \in \mathcal{X}$ such that $y = f(x)$, $y' = f(x')$, and $x \sim x$,. Note that $\mathcal{K}$ satisfies ϵ-d.p. if and only if $\mathcal{H}$ satisfies ϵ-d.p.

If $(\mathcal{Y}, \sim)$ is distance-regular or VT$^+$, then we can apply Theorem 1 to find a bound on the utility. In the following, we assume that the distribution of Y is uniform.

Theorem 3. *Consider a randomized mechanism $\mathcal{H}$, and let y be an element of $\mathcal{Y}$. Assume that $(\mathcal{Y}, \sim)$ is either distance-regular or VT$^+$ and that $\mathcal{H}$ satisfies ϵ-d.p. For each distance d from 0 to the diameter of $(\mathcal{Y}, \sim)$, let n_d be the number of nodes y' at distance d from y. Then we have that:*

$$\mathcal{U}(Y, Z) \leq \frac{1}{\sum_{d} \frac{n_d}{e^{\epsilon\, d}}} \tag{5}$$

The above bound is tight, in the sense that (provided $(\mathcal{Y}, \sim)$ is distance-regular or VT$^+$) we can construct a mechanism $\mathcal{H}$ which satisfies (5) with equality. More precisely, define

$$c = \frac{1}{\sum_{d} \frac{n_d}{e^{\epsilon\, d}}}$$

Then define $\mathcal{H}$ (here identified with its channel matrix for simplicity) as follows:

$$\mathcal{H}_{i,j} = \frac{c}{e^{\epsilon\, d(i,j)}} \tag{6}$$

Theorem 4. *Assume $(\mathcal{Y}, \sim)$ is distance-regular or VT$^+$. Then the matrix $\mathcal{H}$ defined in (6) satisfies ϵ-d.p. and has maximal utility:*

$$\mathcal{U}(Y, Z) = \frac{1}{\sum_d \frac{n_d}{e^{\epsilon\, d}}}$$

Note that we can always define $\mathcal{H}$ as in (6): the matrix so defined will be a legal channel matrix, and it will satisfy ϵ-d.p.. However, if $(\mathcal{Y}, \sim)$ is neither distance-regular nor VT$^+$, then the utility of such $\mathcal{H}$ is not necessarily optimal.

We end this section with an example (borrowed from [1]) to illustrate our technique.

Example 1. Consider a database with electoral information where each row corresponds to a voter and contains the following three fields:

- *Id*: a unique (anonymized) identifier assigned to each voter;
- *City*: the name of the city where the user voted;
- *Candidate*: the name of the candidate the user voted for.

Consider the query *"What is the city with the greatest number of votes for a given candidate cand?"*. For such a query the binary utility function is the natural choice: only the right city gives some gain, and all wrong answers are equally bad. It is easy to see that every two answers are neighbors, i.e. the graph structure of the answers is a clique.

Let us consider the scenario where $City = \{A, B, C, D, E, F\}$ and assume for simplicity that there is a unique answer for the query, i.e., there are no two cities with exactly the same number of individuals voting for candidate *cand*. Table 1 shows two alternative mechanisms providing ϵ-differential privacy (with $\epsilon = \log 2$). The first one, M_1, is based on the truncated geometric mechanism method used in [14] for counting queries (here extended to the case where every pair of answers is neighbor). The second mechanism, M_2, is obtained by applying the definition (6). From Theorem 4 we know that for the uniform input distribution M_2 gives optimal utility.

For the uniform input distribution, it is easy to see that $\mathcal{U}(M_1) = 0.2242 < 0.2857 = \mathcal{U}(M_2)$. Even for non-uniform distributions, our mechanism still provides better utility. For instance, for $p(A) = p(F) = 1/10$ and $p(B) = p(C) = p(D) = P(E) = 1/5$, we have $\mathcal{U}(M_1) = 0.2412 < 0.2857 = \mathcal{U}(M_2)$. This is not too surprising: the geometric mechanism, as well as the Laplacian mechanism proposed by Dwork, perform very well when the domain of answers is provided with a metric and the utility function is not binary[5]. It also works well when $(\mathcal{Y}, \sim)$ has low connectivity, in particular in the cases of a ring and of a line. But in this example, we are not in these cases, because we are considering *binary gain functions* and *high connectivity*.

[5] In the metric case the gain function can take into account the proximity of the reported answer to the real one, the idea being that a close answer, even if wrong, is better than a distant one.

Table 1. Mechanisms for the city with higher number of votes for candidate *cand*

(a) M_1: truncated geometric mechanism

In/Out	A	B	C	D	E	F
A	0.535	0.060	0.052	0.046	0.040	0.267
B	0.465	0.069	0.060	0.053	0.046	0.307
C	0.405	0.060	0.069	0.060	0.053	0.353
D	0.353	0.053	0.060	0.069	0.060	0.405
E	0.307	0.046	0.053	0.060	0.069	0.465
F	0.267	0.040	0.046	0.052	0.060	0.535

(b) M_2: our mechanism

In/Out	A	B	C	D	E	F
A	2/7	1/7	1/7	1/7	1/7	1/7
B	1/7	2/7	1/7	1/7	1/7	1/7
C	1/7	1/7	2/7	1/7	1/7	1/7
D	1/7	1/7	1/7	2/7	1/7	1/7
E	1/7	1/7	1/7	1/7	2/7	1/7
F	1/7	1/7	1/7	1/7	1/7	2/7

7 Related Work

As far as we know, the first work to investigate the relation between differential privacy and information-theoretic leakage *for an individual* was [2]. In this work, a channel is relative to a given database x, and the channel inputs are all possible databases adjacent to x. Two bounds on leakage were presented, one for teh Rényi min entropy, and one for Shannon entropy. Our bound in Proposition 2 is an improvement with respect to the (Rényi min entropy) bound in [2].

Barthe and Köpf [4] were the first to investigates the (more challenging) connection between differential privacy and the Rényi min-entropy leakage *for the entire universe of possible databases*. They consider the "end-to-end differentially private mechanisms", which correspond to what we call $\mathcal{K}$ in our paper, and propose, like we do, to interpret them as information-theoretic channels. They provide a bound for the leakage, but point out that it is not tight in general, and show that there cannot be a domain-independent bound, by proving that for any number of individual u the optimal bound must be at least a certain expression $f(u, \epsilon)$. Finally, they show that the question of providing optimal upper bounds for the leakage of ϵ-differentially private randomized functions in terms of rational functions of ϵ is decidable, and leave the actual function as an open question. In our work we used rather different techniques and found (independently) the same function $f(u, \epsilon)$ (the bound in Theorem 1), but we actually proved that $f(u, \epsilon)$ is the optimal bound[6]. Another difference is that [4] captures the case in which the focus of differential privacy is on hiding *participation* of individuals in a database. In our work, we consider both the participation and the *values* of the participants.

Clarkson and Schneider also considered differential privacy as a case study of their proposal for quantification of integrity [9]. There, the authors analyze database privacy conditions from the literature (such as differential privacy, k-anonymity, and l-diversity) using their framework for utility quantification. In particular, they study the relationship between differential privacy and a notion of leakage (which is different from ours - in particular their definition is based on Shannon entropy) and they provide a tight bound on leakage.

[6] When discussing our result with Barthe and Köpf, they said that they also conjectured that $f(u, \epsilon)$ is the optimal bound.

Heusser and Malacaria [15] were among the first to explore the application of information-theoretic concepts to databases queries. They proposed to model database queries as programs, which allows for statical analysis of the information leaked by the query. However [15] did not attempt to relate information leakage to differential privacy.

In [14] the authors aim at obtaining optimal-utility randomization mechanisms while preserving differential privacy. The authors propose adding noise to the output of the query according to the geometric mechanism. Their framework is very interesting in the sense it provides a general definition of utility for a mechanism M that captures any possible side information and preference (defined as a loss function) the users of M may have. They prove that the geometric mechanism is optimal in the particular case of counting queries. Our results in Section 6 do not restrict to counting queries, but on the other hand we only consider the case of binary loss function.

8 Conclusion and Future Work

In this paper we have investigated the relation between ϵ-differential privacy and leakage, and between ϵ-differential privacy and utility. Our main contribution is the development of a general technique for determining these relations depending on the graph structure induced by the adjacency relation and by the query. We have considered two particular structures, the distance-regular graphs, and the VT^+ graphs, which allow to obtain tight bounds on the leakage and on the utility, and to construct the optimal randomization mechanism satisfying ϵ-differential privacy.

As future work, we plan to extend our result to other kinds of utility functions. In particular, we are interested in the case in which the the answer domain is provided with a metric, and we are interested in taking into account the degree of accuracy of the inferred answer.

References

1. Alvim, M.S., Andrés, M.E., Chatzikokolakis, K., Degano, P., Palamidessi, C.: Differential privacy: on the trade-off between utility and information leakage. Technical report (2011), http://hal.inria.fr/inria-00580122/en/
2. Alvim, M.S., Chatzikokolakis, K., Degano, P., Palamidessi, C.: Differential privacy versus quantitative information flow. Technical report (2010)
3. Andrés, M.E., Palamidessi, C., van Rossum, P., Smith, G.: Computing the leakage of information-hiding systems. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 373–389. Springer, Heidelberg (2010)
4. Barthe, G., Köpf, B.: Information-theoretic bounds for differentially private mechanisms. In: Proc. of CSF (to appear, 2011)
5. Boreale, M., Pampaloni, F., Paolini, M.: Asymptotic information leakage under one-try attacks. In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 396–410. Springer, Heidelberg (2011)

6. Braun, C., Chatzikokolakis, K., Palamidessi, C.: Compositional methods for information-hiding. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 443–457. Springer, Heidelberg (2008)
7. Braun, C., Chatzikokolakis, K., Palamidessi, C.: Quantitative notions of leakage for one-try attacks. In: Proc. of MFPS. ENTCS, vol. 249, pp. 75–91. Elsevier, Amsterdam (2009)
8. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Probability of error in information-hiding protocols. In: Proc. of CSF, pp. 341–354. IEEE, Los Alamitos (2007)
9. Clarkson, M.R., Schneider, F.B.: Quantification of integrity, Tech. Rep. (2011), http://hdl.handle.net/1813/22012
10. Dwork, C.: Differential privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 1–12. Springer, Heidelberg (2006)
11. Dwork, C.: Differential privacy in new settings. In: Proc. of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, pp. 174–183. SIAM, Philadelphia (2010)
12. Dwork, C.: A firm foundation for private data analysis. Communications of the ACM 54(1), 86–96 (2011)
13. Dwork, C., Lei, J.: Differential privacy and robust statistics. In: Proc. of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31-June 2, pp. 371–380. ACM, New York (2009)
14. Ghosh, A., Roughgarden, T., Sundararajan, M.: Universally utility-maximizing privacy mechanisms. In: Proc. of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, pp. 351–360. ACM, New York (2009)
15. Heusser, J., Malacaria, P.: Applied quantitative information flow and statistical databases. In: Degano, P., Guttman, J.D. (eds.) FAST 2009. LNCS, vol. 5983, pp. 96–110. Springer, Heidelberg (2010)
16. Köpf, B., Smith, G.: Vulnerability bounds and leakage resilience of blinded cryptography under timing attacks. In: Proc. of CSF, pp. 44–56. IEEE, Los Alamitos (2010)
17. McIver, A., Meinicke, L., Morgan, C.: Compositional closure for bayes risk in probabilistic noninterference. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 223–235. Springer, Heidelberg (2010)
18. Rényi, A.: On Measures of Entropy and Information. In: Proc. of the 4th Berkeley Symposium on Mathematics, Statistics, and Probability, pp. 547–561 (1961)
19. Smith, G.: On the foundations of quantitative information flow. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 288–302. Springer, Heidelberg (2009)

A 1.488 Approximation Algorithm for the Uncapacitated Facility Location Problem

Shi Li

Department of Computer Science,
Princeton University,
Princeton, NJ, USA, 08540

Abstract. We present a 1.488 approximation algorithm for the metric uncapacitated facility location (UFL) problem. Previously the best algorithm was due to Byrka [1]. By linearly combining two algorithms $A1(\gamma_f)$ for $\gamma_f \approx 1.6774$ and the (1.11,1.78)-approximation algorithm $A2$ proposed by Jain, Mahdian and Saberi [8], Byrka gave a 1.5 approximation algorithm for the UFL problem. We show that if γ_f is randomly selected from some distribution, the approximation ratio can be improved to 1.488. Our algorithm cuts the gap with the 1.463 approximability lower bound by almost 1/3.

Keywords: Approximation, Facility Location Problem, Theory.

1 Introduction

In this paper, we present an improved approximation algorithm for the (metric) uncapacitated facility location (UFL) problem. In the UFL problem, we are given a set of potential facility locations $\mathcal{F}$, each $i \in \mathcal{F}$ with a facility cost f_i, a set of clients $\mathcal{C}$, and a metric d over $\mathcal{F} \cup \mathcal{C}$. The goal is to find a subset $\mathcal{F}' \subset \mathcal{F}$ of locations to open facilities, to minimize the sum of the total facility cost and the connection cost. The total facility cost is $\sum_{i \in \mathcal{F}'} f_i$, and the connection cost is $\sum_{j \in \mathcal{C}} d(j, i_j)$, where i_j is the closest facility in $\mathcal{F}'$ to j.

The UFL problem is NP-hard and has received a lot of attention in the literature. Back to 1982, Hochbaum [6] presented a greedy algorithm with $O(\log n)$ approximation guarantee. Shmoys, Tardos, and Aardal [12] used the filtering technique of Lin and Vitter [10] to give a 3.16 approximation algorithm, which is the first constant approximation. After that, a large number of constant approximation algorithms were proposed([4,9,3,7,8,11]). The current best known approximation ratio for the UFL problem is 1.50, given by Byrka [1].

On the negative side, Guha and Kuller [5] showed that there is no ρ approximation for the UFL problem if $\rho < 1.463$, unless $\mathbf{NP} \subset \mathbf{DTIME}\left(n^{O(\log \log n)}\right)$. Jain et al. [8] generalized the result to that no (λ_f, λ_c)-bifactor approximation exists for $\lambda_c < 1 + 2e^{-\lambda_f}$ unless $\mathbf{NP} \subset \mathbf{DTIME}\left(n^{O(\log \log n)}\right)$. Here, we say that an algorithm is a (λ_f, λ_c)-approximation algorithm if the solution has total cost

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 77–88, 2011.

at most $\lambda_f F^* + \lambda_c C^*$, where F^* and C^* are the facility and the connection cost of an optimal solution, respectively.

Built on the work of Byrka [1], we give a 1.488-approximation algorithm for the UFL problem. Byrka presented an algorithm $A1(\gamma_f)$ which gives the optimal bifactor approximation $(\gamma_f, 1 + 2e^{-\gamma_f})$ for $\gamma_f \geq \gamma_0 \approx 1.6774$. By linearly combining $A1(\gamma_0)$ and the $(1.11, 1.78)$-approximation algorithm $A2$ proposed by Jain, Mahdian and Saberi [8], Byrka was able to give a 1.5 approximation algorithm. We show that if γ_f is randomly selected from some distribution, a linear combination of $A1(\gamma_f)$ and $A2$ can yield a 1.488 approximation.

Due to the hardness result, there is a hard instance for the algorithm $A1(\gamma_f)$ for every λ_f. Roughly speaking, we show that a fixed instance can not be hard for two different λ_f's. Guided by this fact, we first give a bifactor approximation ratio for $A1(\lambda_f)$ that depends on the input instance and then introduce a 0-sum game that characterizes the approximation ratio of our algorithm. The game is between an algorithm player and an adversary. The algorithm player plays a linear combination of $A_1(\lambda_f)$ for a random λ_f and A_2, while the adversary plays an instance. By giving an explicit strategy for the algorithm player, we show that the value of the game is at most 1.488.

We first review the algorithm $A1(\gamma)$ used in [1] in section 2, and then give our improvement in section 3.

2 Review of the Algorithm $A1(\gamma)$ in [1]

In this section, we review the $(\gamma, 1 + 2e^{-\gamma})$-approximation algorithm $A1(\gamma)$ for $\gamma \geq \gamma_0 \approx 1.67736$ in [1].

In $A1(\gamma)$ we first solve the following natural LP for the UFL problem.

$$\begin{aligned}
\min \quad & \sum_{i\in\mathcal{F}, j\in\mathcal{C}} d(i,j)x_{i,j} + \sum_{i\in\mathcal{F}} f_i y_i \quad \text{s.t.} \\
& \sum_{i\in\mathcal{F}} x_{i,j} = 1 \quad \forall j \in \mathcal{C} && (1)\\
& x_{i,j} - y_i \leq 0 \quad \forall i \in \mathcal{F}, j \in \mathcal{C} && (2)\\
& x_{i,j}, y_i \geq 0 \quad \forall i \in \mathcal{F}, j \in \mathcal{C} && (3)
\end{aligned}$$

If the y-variables are fixed, x-variables can be assigned greedily: each $j \in \mathcal{C}$ is connected to 1 fraction of closest facilities. After obtaining a solution (x, y), we modify it by scaling the y-variables up by a constant $\gamma > 1$. Let $\overline{y}$ be the scaled y-variables. We reassign x-variables using the above greedy process to obtain a new solution $(\overline{x}, \overline{y})$. By splitting facilities if necessary, we can assume $x_{i,j} \in \{0, y_i\}$, $\overline{x}_{i,j} \in \{0, \overline{y}_i\}$ and $\overline{y} \leq 1$, for every $i \in \mathcal{C}, j \in \mathcal{F}$.

For some $\mathcal{F}' \subset \mathcal{F}$, define $\mathsf{vol}(\mathcal{F}') = \sum_{i\in\mathcal{F}'} \overline{y}_i$ to be the *volume* of $\mathcal{F}'$. For a client j, we say a facility i is one of his *close facilities* if it fractionally serves j in $(\overline{x}, \overline{y})$. If $\overline{x}_{i,j} = 0$, but i was serving client j in solution (x, y), then we say i is a *distant facility* of client j. Let $\mathcal{F}_j^C, \mathcal{F}_j^D$ to be the set of close facilities, distant facilities of j, respectively. Let $\mathcal{F}_j = \mathcal{F}_j^C \cup \mathcal{F}_j^D$. Define $d_{av}^C(j), d_{av}^D(j), d_{av}(j)$ to be

the average distance from j to $\mathcal{F}_j^C, \mathcal{F}_j^D, \mathcal{F}_j$, respectively. The average distances are with respect to the weights $\overline{y}$ (or equivalently, y). Thus, $d_{av}(j)$ is the connection cost of j in the fractional solution. Define $d_{max}^C(j)$ to be maximum distance from j to a facility in $\mathcal{F}_j^C$. It's easy to see the following facts:

1. $d_{av}^C(j) \leq d_{max}^C(j) \leq d_{av}^D(j), d_{av}^C(j) \leq d_{av}(j) \leq d_{av}^D(j), \forall j \in \mathcal{C}$;
2. $d_{av}(j) = \frac{1}{\gamma} d_{av}^C(j) + \frac{\gamma - 1}{\gamma} d_{av}^D(j), \forall j \in \mathcal{C}$;
3. $\mathsf{vol}(\mathcal{F}_j^C) = 1, \mathsf{vol}(\mathcal{F}_j^D) = \gamma - 1, \mathsf{vol}(\mathcal{F}_j) = \gamma, \forall j \in \mathcal{C}$.

We greedily select a subset of clients $\mathcal{C}'$ in the following way. Initially $\mathcal{C}'' = \mathcal{C}, \mathcal{C}' = \emptyset$. While $\mathcal{C}''$ is not empty, select the client j in $\mathcal{C}''$ with the minimum $d_{av}^C(j) + d_{max}^C(j)$, add j to $\mathcal{C}'$ and remove j and all clients j' satisfying $\mathcal{F}_j^C \cap \mathcal{F}_{j'}^C \neq \emptyset$ from $\mathcal{C}''$. $\mathcal{C}'$ has the following properties :

1. $\mathcal{F}_j^C \cap \mathcal{F}_{j'}^C = \emptyset, \forall j, j' \in \mathcal{C}, j \neq j'$;
2. For every $j \notin \mathcal{C}'$, there exists a $j' \in \mathcal{C}'$ such that $\mathcal{F}_j^C \cap \mathcal{F}_{j'}^C \neq \emptyset$ and $d_{av}^C(j') + d_{max}^C(j') \leq d_{av}^C(j) + d_{max}^C(j)$. This j' is called the *cluster center* of j.

We now randomly round the fractional solution. For each $j \in \mathcal{C}'$, open exactly one of his close facilities randomly with probabilities $\overline{y}_i$. For each facility i that is not a close facility of any client in $\mathcal{C}'$, open it independently with probability $\overline{y}_i$. Each client j is connected to its closest open facility, and let C_j be its connection cost.

It's easy to see that the expected facility cost of the solution is exactly γ times the facility cost in the fractional solution. If $j \in \mathcal{C}'$, $\mathbb{E}[C_j] = d_{av}^C(j) \leq d_{av}(j)$.

Byrka in [1] showed that $\forall j \notin \mathcal{C}'$,

1. The probability that some facility in $\mathcal{F}_j^C$ ($\mathcal{F}_j^D$, resp.) is open is at least $1 - e^{-\mathsf{vol}(\mathcal{F}_j^C)} = 1 - e^{-1}(1 - e^{-\mathsf{vol}(\mathcal{F}_j^D)} = 1 - e^{-(\gamma-1)}$, resp.), and under the condition that the event happens, the expected distance between j and the closest open facility in $\mathcal{F}_j^C$ ($\mathcal{F}_j^D$, resp.) is at most $d_{av}^C(j)$ ($d_{av}^D(j)$, resp.);
2. $d(j, \mathcal{F}_{j'}^C \backslash \mathcal{F}_j) \leq d_{av}^D(j) + d_{max}^C(j) + d_{av}^C(j)$, where j' is cluster center of j; or equivalently, under the condition that there is no open facility in $\mathcal{F}_j$, the expected distance between j and the unique open facility in $\mathcal{F}_{j'}^C$ is at most $d_{av}^D(j) + d_{max}^C(j) + d_{av}^C(j)$.

Since $d_{av}^C(j) \leq d_{av}^D(j) \leq d_{av}^D(j) + d_{max}^C(j) + d_{av}^C(j)$, we have

$$\begin{aligned}
\mathbb{E}[C_j] &\leq (1 - e^{-1}) d_{av}^C(j) + e^{-1}(1 - e^{-(\gamma-1)}) d_{av}^D(j) \\
&\quad + e^{-1} e^{-(\gamma-1)} (d_{av}^D(j) + d_{max}^C(j) + d_{av}^C(j)) \\
&= (1 - e^{-1} + e^{-\gamma}) d_{av}^C(j) + e^{-1} d_{av}^D(j) + e^{-\gamma} d_{max}^C(j) \\
&\leq (1 - e^{-1} + e^{-\gamma}) d_{av}^C(j) + (e^{-1} + e^{-\gamma}) d_{av}^D(j) \qquad (4)
\end{aligned}$$

Notice that the connection cost of j in the fractional solution is $d_{av}(j) = \frac{1}{\gamma} d_{av}^C(j) + \frac{\gamma - 1}{\gamma} d_{av}^D(j)$. We compute the maximum ratio between $(1 - e^{-1} +$

$e^{-\gamma})d_{av}^C(j)+(e^{-1}+e^{-\gamma})d_{av}^D(j)$ and $\frac{1}{\gamma}d_{av}^C(j)+\frac{\gamma-1}{\gamma}d_{av}^D(j)$. Since $d_{av}^C(j) \le d_{av}^D(j)$, the ratio is maximized when $d_{av}^C(j) = d_{av}^D(j) > 0$ or $d_{av}^D(j) > d_{av}^C(j) = 0$. For $\gamma \ge \gamma_0$, the maximum ratio is achieved when $d_{av}^C(j) = d_{av}^D(j) > 0$, in which case the maximum is $1 + 2e^{-\gamma}$. Thus, the algorithm $A1(\gamma_0)$ gives a $(\gamma_0 \approx 1.67736, 1 + 2e^{-\gamma_0} \approx 1.37374)$ bifactor approximation.[1]

3 A 1.488 Approximation Algorithm for the UFL Problem

In this section, we give our approximation algorithm for the UFL problem. Our algorithm is also based on the combination of the $A1(\gamma)$ and $A2$. However, instead of using $A1(\gamma)$ for a fixed γ, we randomly select γ from some distribution.

To understand why this approach can reduce the approximation ratio, we list all requirements that the upper bound in (4) is tight.

1. The facilities in $\mathcal{F}_j$ have tiny weights. In other words, $\max_{i\in\mathcal{F}_j} \overline{y}_i$ tends to 0. Moreover, all these facilities were independently sampled in the algorithm. These conditions are necessary to tighten the $1 - e^{-1}$ ($1 - e^{-(\gamma-1)}$ resp.) upper bound for the probability that at least 1 facility in $\mathcal{F}_j^C$ ($\mathcal{F}_j^D$ resp.) is open.
2. The distances from j to all the facilities in $\mathcal{F}_j^C$ ($\mathcal{F}_j^D$ resp.) are the same. Otherwise, the expected distance from j to the closest open facility in $\mathcal{F}_j^C$ ($\mathcal{F}_j^D$ resp.), under the condition that it exists, is strictly smaller than $d_{av}^C(j)$ ($d_{av}^D(j)$ resp.).
3. $d_{max}^C(j) = d_{av}^D(j)$. This is also required since we used $d_{av}^D(j)$ as an upper bound of $d_{max}^C(j)$ to get (4).

To satisfy all the above conditions, the distances from j to $\mathcal{F}_j$ must be distributed as follows. $1/(\gamma + \epsilon)$ fraction of facilities in $\mathcal{F}_j$ have distances a to j, and the other $1 - 1/(\gamma + \epsilon)$ fraction have distances $b \ge a$ to j. For ϵ tending to 0, $d_{av}^C(j) = a$ and $d_{max}^C(j) = d_{av}^D(j) = b$.

As discussed earlier, if $a = b$, then $\mathbb{E}[C_j]/d_{av}(j) \le 1 + 2e^{-\gamma}$. Intuitively, the bad cases should have $a \ll b$. However, if we replace γ with $\gamma + 1.01\epsilon$ (say), we have $d_{max}^C(j)$ equals $d_{av}^C(j) = a$, instead of $d_{av}^D(j) = b$ for the above distribution. Thus, we can greatly reduce the approximation ratio if the distributions for all j's are of the above form.

Hence, using only two different γ's, we are already able to make an improvement. To give a better analysis, we first give a upper bound on $\mathbb{E}[C_j]$, in terms of the distribution of distances from j to $\mathcal{F}_j$, not just $d_{av}^C(j)$ and $d_{av}^D(j)$, and then give an explicit distribution for γ by introducing a 0-sum game.

[1] Byrka's analysis in [1] was a little bit different; it used some variables from the dual LP. Later in [2], Byrka et al. gave an analysis without using the dual LP, which is the one we cite in our paper.

3.1 Upper-Bounding the Expected Connection Cost of a Client

We bound $\mathbb{E}[C_j]$ in this subsection. It suffices to assume $j \notin \mathcal{C}'$, since we can think of a client $j \in \mathcal{C}'$ as a client $j \notin \mathcal{C}'$ which has a co-located client $j' \in \mathcal{C}'$. Similar to [1], we give an upper bound on $d(j, \mathcal{F}_{j'}^C \backslash \mathcal{F}_j)$. The bound and the proof are the same as the counterparts in [1], except that we made a slight improvement. The improvement is not essential to the final approximation ratio; however, it will simply the analytical proof in subsection 3.2.

Lemma 1. *For some $j \notin \mathcal{C}'$, let j' be the cluster center of j. So $j' \in \mathcal{C}', \mathcal{F}_j^C \cap \mathcal{F}_{j'}^C \neq \emptyset$ and $d_{av}^C(j') + d_{max}^C(j') \leq d_{av}^C(j) + d_{max}^C(j)$. We have, for any $\gamma \geq 1$,*

$$d(j, \mathcal{F}_{j'}^C \backslash \mathcal{F}_j) \leq (2-\gamma) d_{max}^C(j) + (\gamma - 1) d_{av}^D(j) + d_{max}^C(j') + d_{av}^C(j'). \quad (5)$$

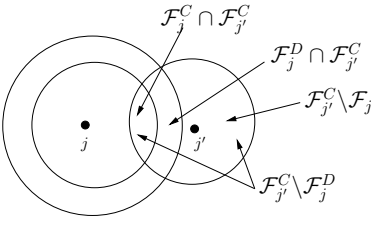


Fig. 1. Sets of facilities used in the proof

Proof. Fig. 1 illustrates the sets of facilities we are going to use in the proof. If the $d(j, j') \leq (2-\gamma) d_{max}^C(j) + (\gamma - 1) d_{av}^D(j) + d_{av}^C(j')$, the remaining $d_{max}^C(j')$ is enough for the distance between j' and any facility in $\mathcal{F}_{j'}^C$. So, we will assume

$$d(j, j') \geq (2-\gamma) d_{max}^C(j) + (\gamma - 1) d_{av}^D(j) + d_{av}^C(j'). \quad (6)$$

(6) implies $d(j, j') \geq d_{max}^C(j) + d_{av}^C(j')$. Since $d(j, \mathcal{F}_j^C \cap \mathcal{F}_{j'}^C) \leq d_{max}^C(j)$, we have $d(j', \mathcal{F}_j^C \cap \mathcal{F}_{j'}^C) \geq d_{av}^C(j')$. If $d(j', \mathcal{F}_j^D \cap \mathcal{F}_{j'}^C) \geq d_{av}^C(j')$, then $d(j', \mathcal{F}_{j'}^C \backslash \mathcal{F}_j) \leq d_{av}^C(j')$ and the lemma follows from the fact that $d(j, j') \leq d_{max}^C(j) + d_{max}^C(j') \leq (2-\gamma) d_{max}^C(j) + (\gamma - 1) d_{av}^D(j) + d_{max}^C(j')$.

So, we can also assume

$$d(j', \mathcal{F}_j^D \cap \mathcal{F}_{j'}^C) = d_{av}^C(j') - z \quad (7)$$

for some positive z. Let $\hat{y} = \mathsf{vol}(\mathcal{F}_j^D \cap \mathcal{F}_{j'}^C)$. Notice that $\hat{y} \leq \max\{\gamma - 1, 1\}$. (7) implies

$$d(j', \mathcal{F}_{j'}^C \backslash \mathcal{F}_j^D) = d_{av}^C(j') + \frac{\hat{y}}{1-\hat{y}} z \quad (8)$$

From (6) and (7), we get

$$\begin{aligned} d(j, \mathcal{F}_j^D \cap \mathcal{F}_{j'}^C) &\geq (2-\gamma) d_{max}^C(j) + (\gamma - 1) d_{av}^D(j) + z & (9) \\ &= d_{av}^D(j) - (2-\gamma)(d_{av}^D(j) - d_{max}^C(j)) + z & (10) \end{aligned}$$

This further implies

$$\begin{aligned} d^C_{max}(j) &\le d(j, \mathcal{F}^D_j \backslash \mathcal{F}^C_{j'}) \\ &\le d^D_{av}(j) - \frac{\hat{y}}{\gamma - 1 - \hat{y}} \left(z - (2-\gamma)(d^D_{av}(j) - d^C_{max}(j)) \right) \\ d^D_{av}(j) - d^C_{max}(j) &\ge \frac{\hat{y}}{\gamma - 1 - \hat{y}} \left(z - (2-\gamma)(d^D_{av}(j) - d^C_{max}(j)) \right) \\ d^D_{av}(j) - d^C_{max}(j) &\ge \frac{\hat{y}}{\gamma - 1 - \hat{y}} z \Big/ \left(1 + \frac{(2-\gamma)\hat{y}}{\gamma - 1 - \hat{y}} \right) = \frac{\hat{y} z}{(\gamma - 1)(1 - \hat{y})} \end{aligned} \tag{11}$$

Notice that we used $1 + \frac{(2-\gamma)\hat{y}}{\gamma - 1 - \hat{y}} \ge 0$. From (6) and (11), we get

$$\begin{aligned} d(j', \mathcal{F}^C_j \cap \mathcal{F}^C_{j'}) &\ge d(j, j') - d(j, \mathcal{F}^C_j \cap \mathcal{F}^C_{j'}) \\ &\ge (2-\gamma) d^C_{max}(j) + (\gamma - 1) d^D_{av}(j) + d^C_{av}(j') - d^C_{max}(j) \\ &= (\gamma - 1)(d^D_{av}(j) - d^C_{max}(j)) + d^C_{av}(j') \\ &\ge \frac{\hat{y}}{1 - \hat{y}} z + d^C_{av}(j') \end{aligned} \tag{12}$$

Combining the above inequality and (8), we have

$$d(j', \mathcal{F}^C_{j'} \backslash \mathcal{F}_j) \le d^C_{av}(j') + \frac{\hat{y}}{1 - \hat{y}} z. \tag{13}$$

So,

$$\begin{aligned} d(j, \mathcal{F}^C_{j'} \backslash \mathcal{F}_j) &\le d^C_{max}(j) + d^C_{max}(j') + d(j', \mathcal{F}^C_{j'} \backslash \mathcal{F}_j) \\ &\le (2-\gamma) d^C_{max}(j) + (\gamma - 1) \left(d^D_{av}(j) - \frac{\hat{y} z}{(\gamma - 1)(1 - \hat{y})} \right) \\ &\quad + d^C_{max}(j') + d^C_{av}(j') + \frac{\hat{y}}{1 - \hat{y}} \\ &= (2-\gamma) d^C_{max}(j) + (\gamma - 1) d^D_{av}(j) + d^C_{max}(j') + d^C_{av}(j') \end{aligned}$$

Lemma 2

$$d(j, \mathcal{F}^C_{j'} \backslash \mathcal{F}_j) \le \gamma d_{av}(j) + (3 - \gamma) d^C_{max}(j). \tag{14}$$

Proof. Noticing that $d^C_{max}(j') + d^C_{av}(j') \le d^C_{max}(j) + d^C_{av}(j)$, the proof is straightforward.

$$\begin{aligned} d(j, \mathcal{F}^C_{j'} \backslash \mathcal{F}_j) &\le (2-\gamma) d^C_{max}(j) + (\gamma - 1) d^D_{av}(j) + d^C_{max}(j') + d^C_{av}(j') \\ &\le (2-\gamma) d^C_{max}(j) + (\gamma - 1) d^D_{av}(j) + d^C_{max}(j) + d^C_{av}(j) \\ &= \gamma \left(\frac{1}{\gamma} d^C_{av}(j) + \frac{\gamma - 1}{\gamma} d^D_{av}(j) \right) + (3 - \gamma) d^C_{max}(j) \\ &= \gamma d_{av}(j) + (3 - \gamma) d^C_{max}(j) \end{aligned}$$

For a client $j \in \mathcal{C}$, define $h_j : [0,1] \to \mathbb{R}^*$ to be the distribution of distances form j to $\mathcal{F}_j$ in the following way. Let $i_1, i_2, \cdots, i_m$ the facilities in $\mathcal{F}_j$, in the non-decreasing order of distances to j. Then $h_j(p) = d_{i_t,j}$, where t is the minimum number such that $\sum_{s=1}^{t} y_{i_s} \geq p$. Notice that h_j is defined using the y, not $\overline{y}$, and is thus independent of γ. Define $h(p) = \sum_{j \in \mathcal{C}} h_j(p)$. Observe that h_js and h are non-decreasing functions. Furthermore,

$$d_{av}(j) = \int_0^1 h_j(p)\mathbf{d}p, \qquad d_{av}^C(j) = \gamma \int_0^{1/\gamma} h_j(p)\mathbf{d}p,$$
$$d_{max}^C(j) = h_j(1/\gamma), \qquad d_{av}^D(j) = \frac{\gamma}{\gamma - 1} \int_{1/\gamma}^1 h_j(p)\mathbf{d}p.$$

Lemma 3. *For any client j,*

$$\mathbb{E}[C_j] \leq \int_0^1 h_j(p)e^{-\gamma p}\gamma \mathbf{d}p + e^{-\gamma}\left(\gamma \int_0^1 h_j(p)\mathbf{d}p + (3-\gamma)h_j\left(\frac{1}{\gamma}\right)\right). \quad (15)$$

Proof. Let $j' \in \mathcal{C}'$ be the cluster center of j. We connect j to the closest open facility in $\mathcal{F}_j \cup \mathcal{F}_{j'}^C$.

We can assume that facilities in $\mathcal{F}_j \backslash \mathcal{F}_{j'}^C$ are independently sampled; otherwise, $\mathbb{E}[C_j]$ can only be smaller. Indeed, consider two distributions p_1 and p_2 whose only difference is the following. In p_1 two facilities i and i' are dependently sampled (with probability $\overline{y}_i$, i is open, with probability $\overline{y}_{i'}$, i' is open, and with probability $1-\overline{y}_i-\overline{y}_{i'}$, none of them are open), while in p_2 they are independently sampled. W.L.O.G, assume $d(j,i) \leq d(j,i')$. We consider the distribution of the distance from j to the closest open facility in $\{i, i'\}$ (∞ if it does not exist). In p_1, the distribution is: with probability $\overline{y}_i$, we get $d(j,i)$; with probability $\overline{y}_{i'}$, we get $d(j,i')$ and with the remaining $1 - \overline{y}_i - \overline{y}_{i'}$ probability, we get ∞. In p_2, the distribution is: with probability $\overline{y}_i$, we get $d(j,i)$, with probability $(1-\overline{y}_i)\overline{y}_{i'}$, we get $d(j,i')$ and with the remaining probability $(1-\overline{y}_i)(1-\overline{y}_{i'})$, we get ∞. So, the distribution for p_2 strictly dominates the distribution for p_1. The expected connection cost w.r.t p_2 is at least as large as the expected connection cost w.r.t p_1. This argument can be easily extended to more than 2 facilities.

Then, we perform the following sequence of operations:

1. Split the set $\mathcal{F}_{j'}^C$ into two subsets: $\mathcal{F}_{j'}^C \cap \mathcal{F}_j$, and $\mathcal{F}_{j'}^C \backslash \mathcal{F}_j$;
2. Scale up $\bar{y}$ values in $\mathcal{F}_{j'}^C \backslash \mathcal{F}_j$ so that the volume of $\mathcal{F}_{j'}^C \backslash \mathcal{F}_j$ becomes 1;
3. Assume $\mathcal{F}_{j'}^C \cap \mathcal{F}_j$ and $\mathcal{F}_{j'}^C \backslash \mathcal{F}_j$ are independently sampled.

We show that the sequence of operations does not change $\mathbb{E}[C_j]$. Indeed, consider distribution of the distance between j and the closest open facility in $\mathcal{F}_{j'}^C$. The distribution does not change after we performed the operations, since $d_{max}(j, \mathcal{F}_{j'}^C \cap \mathcal{F}_j) \leq d_{min}(j, \mathcal{F}_{j'}^C \backslash \mathcal{F}_j)$, where d_{max} and d_{min} denotes the maximum and the minimum distance from a client to a set of facilities, respectively.

Again, we can pretend that facilities in $\mathcal{F}_{j'}^C \cap \mathcal{F}_j$ are independently sampled. Now, we are in a situation where, facilities in $\mathcal{F}_j$ are independently sampled, exact 1 facility in $\mathcal{F}_{j'}^C \backslash \mathcal{F}_j$ is open, with probabilities proportional to the $\overline{y}$ values.

We split each facility $i \in \mathcal{F}_j$ into facilities with infinitely small $\overline{y}$ values. This can only increase $\mathbb{E}[C_j]$. Thus,

$$\begin{aligned}\mathbb{E}[C_j] &\leq \int_0^1 h_j(p)e^{-\gamma p}\gamma \mathbf{d}p + e^{-\gamma} d(j, \mathcal{F}_{j'}^C \backslash \mathcal{F}_j) \\ &\leq \int_0^1 h_j(p)e^{-\gamma p}\gamma \mathbf{d}p + e^{-\gamma}\left(\gamma \int_0^1 h_j(p)\mathbf{d}p + (3-\gamma)h_j\left(\frac{1}{\gamma}\right)\right).\end{aligned}$$

Lemma 4. *The expected connection cost of the integral solution is*

$$\mathbb{E}[C] \leq \int_0^1 h(p)e^{-\gamma p}\gamma \mathbf{d}p + e^{-\gamma}\left(\gamma \int_0^1 h(p)\mathbf{d}p + (3-\gamma)h\left(\frac{1}{\gamma}\right)\right). \tag{16}$$

Proof. Summing up (15) over all clients j will give us the lemma.

3.2 An Explicit Distribution of γ

In this subsection, we give an explicit distribution for γ by introducing a 0-sum game. For fixed h and γ, let's define

$$\alpha(\gamma, h) = \int_0^1 h(p)e^{-\gamma p}\gamma \mathbf{d}p + e^{-\gamma}\left(\gamma \int_0^1 h(p)\mathbf{d}p + (3-\gamma)h\left(\frac{1}{\gamma}\right)\right). \tag{17}$$

We can scale h so that $\int_0^1 h(p)\mathbf{d}p = 1$. Then,

$$\alpha(\gamma, h) = \int_0^1 h(p)e^{-\gamma p}\gamma \mathbf{d}p + e^{-\gamma}\left(\gamma + (3-\gamma)h\left(\frac{1}{\gamma}\right)\right). \tag{18}$$

We consider a 0-sum game between an algorithm player A and an adversary B. The strategy of player A is a pair (μ, θ), where $0 \leq \theta \leq 1$ and μ is $1-\theta$ times a probability density function for γ. $\theta + \int_1^\infty \mu(\gamma)\mathbf{d}\gamma = 1$. It corresponds to running $A2$ with probability θ and running $A1(\gamma)$ with probability $\mu(\gamma)\mathbf{d}\gamma$. The strategy of player B is a monotone non-negative function h over $[0,1]$ such that $\int_0^1 h(p)\mathbf{d}p = 1$. The value of the game is

$$\nu(\mu, \theta, h) = \max\left\{\int_1^2 \gamma\mu(\gamma)\mathbf{d}\gamma + 1.11\theta, \int_1^2 \alpha(\gamma, h)\mu(\gamma)\mathbf{d}\gamma + 1.78\theta\right\}. \tag{19}$$

Then, our goal becomes finding a strategy for A that minimizes the value. For a fixed strategy (θ, μ) of player A, the best strategy of player B is a threshold function h_q, for some $0 \leq q < 1$, where

$$h_q(p) = \begin{cases} 0 & p < q \\ \frac{1}{1-q} & p \geq q \end{cases} \tag{20}$$

To obtain the value of this game, we discretize the domain [1,3) for μ into many small intervals divided by points $\{\gamma_i = 1 + i/n : 0 \leq i \leq 2n\}$. Thus, the value of the game is approximately characterized by the following LP.

$$\min \quad \beta \qquad \text{s.t}$$

$$\frac{1}{n}\sum_{i=1}^{2n} x_i + \theta \geq 1 \tag{21}$$

$$\frac{1}{n}\sum_{i=1}^{2n} \frac{\gamma_{i-1}+\gamma_i}{2} x_i + 1.11\theta \leq \beta \tag{22}$$

$$\frac{1}{n}\sum_{i=1}^{2n} \alpha\left(\frac{\gamma_{i-1}+\gamma_i}{2}, h_q\right) x_i + 1.78\theta \leq \beta \quad \forall q \in [0,1) \tag{23}$$

$$x_1, x_2, \cdots, x_{2n}, \theta \geq 0 \tag{24}$$

We discretize the domain [0,1) for q and solve the above LP for $n = 500$ using Matlab. We get a mixed strategy for player A that achieves value 1.4879. It is roughly of the following form. $\theta \approx 0.2$; $\gamma \approx 1.5$ with probability 0.5; the remaining 0.3 probability is distributed between $\gamma \approx 1.5$ and $\gamma \approx 2$.

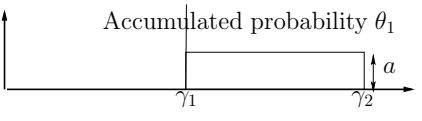


Fig. 2. The distribution of γ

In light of the program generated solution, we give a pure analytical strategy of player A and show that the value of the game is at most 1.488. With probability θ_2, we run $A2$; with probability θ_1, we run $A1(\gamma)$ with $\gamma = \gamma_1$; with probability $1 - \theta_2 - \theta_1$, we run $A1(\gamma)$ with γ randomly chosen between γ_1 and γ_2. So, the function μ is

$$\mu(\gamma) = \theta_1 \delta(\gamma - \gamma_1) + a I_{\gamma_1,\gamma_2}(\gamma), \tag{25}$$

where δ is the Dirac-Delta function, $a = \dfrac{1-\theta_1-\theta_2}{\gamma_2-\gamma_1}$, and $I_{\gamma_1,\gamma_2}(\gamma)$ is 1 if $\gamma_1 < \gamma < \gamma_2$ and 0 otherwise(See Fig. 2). The values of $\theta_1, \theta_2, \gamma_1, \gamma_2, a$ is decided later.

The scaling factor for the facility cost is

$$\lambda_f = \theta_1\lambda_1 + a(\gamma_2-\gamma_1)\frac{\gamma_1+\gamma_2}{2} + 1.11\theta_2. \tag{26}$$

Now, we consider the scaling factor λ_c when $h = h_q$.

$$\begin{aligned}
\lambda_c(q) &= \int_1^\infty \left(\int_0^1 e^{-\gamma p}\gamma h_q(p)\mathbf{d}p + e^{-\gamma}(\gamma+(3-\gamma)h_q(1/\gamma))\right)\mu(\gamma)\mathbf{d}\gamma + 1.78\theta_2 \\
&= \int_{\gamma_1}^{\gamma_2}\int_q^1 e^{-\gamma p}\gamma\frac{1}{1-q}\mathbf{d}pa\mathbf{d}\gamma + \int_{\gamma_1}^{\gamma_2} e^{-\gamma}\gamma a\mathbf{d}\gamma + \int_{\gamma_1}^{\gamma_2}(3-\gamma)h_q(1/\gamma)a\mathbf{d}\gamma \\
&\quad + \theta_1\int_q^1 e^{-\gamma_1 p}\gamma_1\frac{1}{1-q}\mathbf{d}p + \theta_1 e^{-\gamma_1}(\gamma_1+(3-\gamma_1)h_q(1/\gamma_1)) + 1.78\theta_2 \\
&= B_1(q) + B_2(q) + B_3(q) + 1.78\theta_2,
\end{aligned} \tag{27}$$

where

$$\begin{aligned}
B_1(q) &= \int_{\gamma_1}^{\gamma_2}\int_q^1 e^{-\gamma p}\gamma\frac{1}{1-q}\mathbf{d}pa\mathbf{d}\gamma + \int_{\gamma_1}^{\gamma_2} e^{-\gamma}\gamma a\mathbf{d}\gamma \\
&= \frac{a}{1-q}\int_{\gamma_1}^{\gamma_2}(e^{-\gamma q} - e^{-\gamma})\mathbf{d}\gamma - a(\gamma+1)e^{-\gamma}\big|_{\gamma_1}^{\gamma_2} \\
&= \frac{a}{(1-q)q}(e^{-\gamma_1 q} - e^{-\gamma_2 q}) - \frac{a}{1-q}(e^{-\gamma_1} - e^{-\gamma_2}) \\
&\quad + a((\gamma_1+1)e^{-\gamma_1} - (\gamma_2+1)e^{-\gamma_2}),
\end{aligned} \tag{28}$$

$$\begin{aligned}
B_2(q) &= \int_{\gamma_1}^{\gamma_2}(3-\gamma)h_q(1/\gamma)a\mathbf{d}\gamma \\
&= \begin{cases} \frac{a}{1-q}\left((2-\gamma_1)e^{-\gamma_1} - (2-\gamma_2)e^{-\gamma_2}\right) & 0 \le q < 1/\gamma_2 \\ \frac{a}{1-q}\left((2-\gamma_1)e^{-\gamma_1} - (2-1/q)e^{-1/q}\right) & 1/\gamma_2 \le q \le 1/\gamma_1 \\ 0 & 1/\gamma_1 < q < 1 \end{cases},
\end{aligned} \tag{29}$$

$$\begin{aligned}
B_3(q) &= \theta_1\int_q^1 e^{-\gamma_1 p}\gamma_1\frac{1}{1-q}\mathbf{d}p + \theta_1 e^{-\gamma_1}(\gamma_1+(3-\gamma_1)h_q(1/\gamma_1)) \\
&= \theta_1\left(\frac{1}{1-q}(e^{-\gamma_1 q} - e^{-\gamma_1}) + e^{-\gamma_1}\gamma_1 + e^{-\gamma_1}(3-\gamma_1)h_q(1/\gamma_1)\right) \\
&= \begin{cases} \theta_1\left(\frac{1}{1-q}(e^{-\gamma_1 q} - e^{-\gamma_1}) + e^{-\gamma_1}\gamma_1 + \frac{e^{-\gamma_1}(3-\gamma_1)}{1-q}\right) & 0 \le q \le 1/\gamma_1 \\ \theta_1\left(\frac{1}{1-q}(e^{-\gamma_1 q} - e^{-\gamma_1}) + e^{-\gamma_1}\gamma_1\right) & 1/\gamma_1 < q < 1 \end{cases}.
\end{aligned} \tag{30}$$

So, we have 3 cases :

1. $0 \leq q < 1/\gamma_2$

$$\lambda_c(q) = \frac{a}{(1-q)q}(e^{-\gamma_1 q} - e^{-\gamma_2 q}) + \frac{A_1}{1-q} + \theta_1 \frac{e^{-\gamma_1 q}}{1-q} + A_2.$$
$$\text{where } A_1 = a(e^{-\gamma_1} - \gamma_1 e^{-\gamma_1} - e^{-\gamma_2} + \gamma_2 e^{-\gamma_2}) + 2\theta_1 e^{-\gamma_1} - \theta_1 e^{-\gamma_1}\gamma_1$$
$$A_2 = a((\gamma_1+1)e^{-\gamma_1} - (\gamma_2+1)e^{-\gamma_2}) + \theta_1 e^{-\gamma_1}\gamma_1 + 1.78\theta_2. \quad (31)$$

2. $1/\gamma_2 \leq q \leq 1/\gamma_1$

$$\lambda_c(q) = \frac{a}{(1-q)q}(e^{-\gamma_1 q} - e^{-\gamma_2 q}) + \frac{A_1}{1-q} + \theta_1 \frac{e^{-\gamma_1 q}}{1-q} + A_2 + \frac{a}{1-q}\left((2-\gamma_2)e^{-\gamma_2} - (2-1/q)e^{-1/q}\right). \quad (32)$$

3. $1/\gamma_1 < q < 1$

$$\lambda_c(q) = \frac{a}{(1-q)q}(e^{-\gamma_1 q} - e^{-\gamma_2 q}) + \frac{A_3}{1-q} + \theta_1 \frac{e^{-\gamma_1 q}}{1-q} + A_2, \quad (33)$$
$$\text{where } A_3 = a(-e^{-\gamma_1} + e^{-\gamma_2}) - \theta_1 e^{-\gamma_1}. \quad (34)$$

We set $\gamma_1 = 1.479311, \gamma_2 = 2.016569, \theta_1 = 0.503357, \theta_2 \approx 0.195583, a = 0.560365$. We have

$$\lambda_f = \theta_1\lambda_1 + a(\gamma_2 - \gamma_1)\frac{\gamma_1+\gamma_2}{2} + 1.11\theta_2 \approx 1.487954. \quad (35)$$

$\lambda_c(q)$ has the maximum value about 1.487989, achieved at $q = 0$(see Fig. 3).

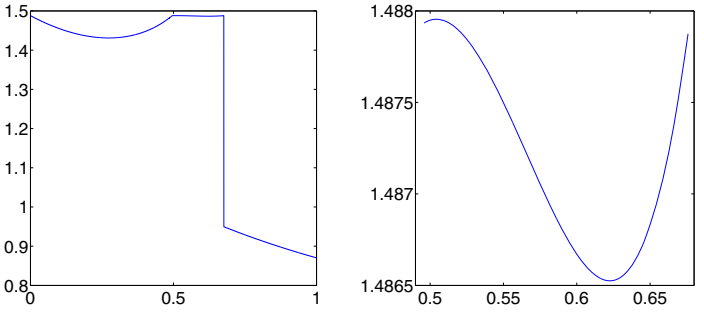


Fig. 3. The function $\lambda_c(q)$. The curve on the right-hand side is the function restricted to the interval $(1/\gamma_2, 1/\gamma_1)$.

Thus, we get a (1.488, 1.488)-bifactor approximation for the UFL problem.

Acknowledgments. I thank Moses Charikar, my advisor, for helpful discussions. I also thank the anonymous reviews for their comments and suggestions.

References

1. Byrka, J.: An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) RANDOM 2007 and APPROX 2007. LNCS, vol. 4627, pp. 29–43. Springer, Heidelberg (2007)
2. Byrka, J., Ghodsi, M., Srinivasan, A.: Lp-rounding algorithms for facility-location problems. in arxiv1007.3611 (2010)
3. Charikar, M., Guha, S.: Improved combinatorial algorithms for the facility location and k-median problems. In: Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, pp. 378–388 (1999)
4. Chudak, F.A., Shmoys, D.B.: Improved approximation algorithms for the uncapacitated facility location problem. SIAM J. Comput. 33(1), 1–25 (2004)
5. Guha, S., Khuller, S.: Greedy strikes back: Improved facility location algorithms. Journal of Algorithms, 649–657 (1998)
6. Hochbaum, D.S.: Heuristics for the fixed cost median problem. Mathematical Programming 22, 148–162 (1982)
7. Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.V.: Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. J. ACM 50, 795–824 (2003), http://doi.acm.org/10.1145/950620.950621
8. Jain, K., Mahdian, M., Saberi, A.: A new greedy approach for facility location problems. In: Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing, STOC 2002, pp. 731–740. ACM, New York (2002), http://doi.acm.org/10.1145/509907.510012
9. Korupolu, M.R., Plaxton, C.G., Rajaraman, R.: Analysis of a local search heuristic for facility location problems. In: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1998, Philadelphia, PA, USA, pp. 1–10 (1998), http://portal.acm.org/citation.cfm?id=314613.314616
10. Lin, J., Vitter, J.S.: Approximation algorithms for geometric median problems. Inf. Process. Lett. 44, 245–249 (1992), http://portal.acm.org/citation.cfm?id=152566.152569
11. Mahdian, M., Ye, Y., Zhang, J.: Approximation algorithms for metric facility location problems. SIAM J. Comput. 36(2), 411–432 (2006)
12. Shmoys, D.B., Tardos, E., Aardal, K.: Approximation algorithms for facility location problems (extended abstract). In: STOC 1997: Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing, pp. 265–274. ACM, New York (1997)

Rice's Theorem for μ-Limit Sets of Cellular Automata*

Martin Delacourt

Laboratoire d'Informatique Fondamentale de Marseille
Université de Provence, France

Abstract. Cellular automata are a parallel and synchronous computing model, made of infinitely many finite automata updating according to the same local rule. Rice's theorem states that any nontrivial property over computable functions is undecidable. It has been adapted by Kari to limit sets of cellular automata [7], that is the set of configurations that can be reached arbitrarily late. This paper proves a new Rice theorem for μ-limit sets, which are sets of configurations often reached arbitrarily late.

1 Introduction

In the field of decidability, a major result is the Rice theorem [10] which states that every property over computable functions is either trivial or undecidable. This being stated for computable functions, it is quite natural to expect a similar result for other computational systems.

In this paper, we will focus on cellular automata, a massively parallel model of computation introduced by von Neumann [9]. Cellular automata are composed of infinitely many cells that evolve synchronously following the same local rule. The dynamics of these objects have been well studied, and in particular the notion of limit set i.e. the set of configurations that can be seen arbitrarily late. An important step was achieved by Kari with the equivalent of Rice's theorem for limit sets [7].

Another point of view is to look at configurations that can appear arbitrarily late and often. This supposes to choose the initial configuration according to a measure μ. This approach led to μ-attractors [6] and then to μ-limit sets introduced in [8]. Configurations of the μ-limit set are those containing only patterns whose probability does not tend to 0, or equivalently configurations obtained starting from a random initial configuration.

Some results on μ-limit sets are already known, such as the undecidability of the μ-nilpotency [2]. We deal here with Rice's theorem for μ-limit sets. With a construction similar to the one presented in [1] to obtain complex subshifts as μ-limit sets, we will reduce any nontrivial property to the question of μ-nilpotency. First we give some definitions, then two sections are devoted to the construction of an appropriate cellular automaton, and finally we will be able to prove the reduction.

* Thanks to the project ANR EMC: ANR-09-BLAN-0164.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 89–100, 2011.

2 Definitions

2.1 Words and Density

For a finite set Q called an *alphabet*, denote $Q^* = \bigcup_{n\in\mathbb{N}} Q^n$ the set of all finite words over Q. The *length* of $u = u_0u_1 \dots u_{n-1}$ is $|u| = n$. We denote $Q^{\mathbb{Z}}$ the set of *configurations* over Q, which are mappings from $\mathbb{Z}$ to Q, and for $c \in Q^{\mathbb{Z}}$, we denote c_z the image of $z \in \mathbb{Z}$ by c. We define *semi-configurations* the same way as elements of $Q^{\mathbb{N}}$. When there is no ambiguity, we will speak of configurations for *configurations* or *semi-configurations* indistinctly. For $u \in Q^*$ and $0 \leq i \leq j < |u|$ we define the *subword* $u_{[i,j]} = u_iu_{i+1} \dots u_j$; this definition can be extended to a configuration $c \in Q^{\mathbb{Z}}$ as $c_{[i,j]} = c_ic_{i+1} \dots c_j$ for $i, j \in \mathbb{Z}$ with $i \leq j$. The *language* of $S \subset Q^{\mathbb{Z}}$ is defined by

$$L(S) = \{u \in Q^* : \exists c \in S,\ \exists i \in \mathbb{Z} \text{ such that } u = c_{[i,i+|u|-1]}\}.$$

For every $u \in Q^*$ and $i \in \mathbb{Z}$, we define the *cylinder* $[u]_i$ as the set of configurations containing the word u in position i that is to say $[u]_i = \{c \in Q^{\mathbb{Z}} : c_{[i,i+|u|-1]} = u\}$. If the cylinder is at the position 0, we just denote it by $[u]$.

For all $u, v \in Q^*$ define $|v|_u$ the *number of instances* of u in v as:

$$|v|_u = \text{card}\{i \in [0, |v| - |u|] : v_{[i,i+|u|-1]} = u\}$$

For finite words $u, v \in Q^*$, if $|u| < |v|$, the density of u in v is defined as $d_v(u) = \frac{|v|_u}{|v|-|u|+1}$. For a configuration $c \in Q^{\mathbb{Z}}$, the *density* $d_c(v)$ of a finite word v is:

$$d_c(v) = \limsup_{n\to+\infty} \frac{|c_{[-n,n]}|_v}{2n + 2 - |v|}.$$

These definitions can be generalized for a set of words $W \subset Q^*$ (in which no word is prefix of another one), we note $|u|_W$ and $d_c(W)$. We can give similar definitions for semi-configurations (indexed by $\mathbb{N}$) too.

Definition 1 (Generic configuration). *A configuration c is said to be* generic *for an alphabet Q if there exists a constant e, such that, for any two words $|u| = |v|$ in Q^*, we have $\frac{d_c(u)}{d_c(v)} \leq e$. If, moreover, any word of length k has density $\frac{1}{|Q|^k}$, the configuration is said to be* normal.

In this paper, we will use a particular generic configuration, that we define here.

Definition 2 (de Bruijn sequence). *A* de Bruijn sequence *of order $n \in \mathbb{N}$ over an alphabet Q is a word of length $|Q|^n + n - 1$ that contains every word of length n as a subword.*

Thanks to [4], we know that there exists a specific de Bruijn sequence of order n noted $DB(n)$ produced with space $O(n)$ and time $O(|Q|^n)$. We will use this sequence in what follows.

Definition 3 (de Bruijn configuration). *The* de Bruijn configuration c_{DB} *is the concatenation of de Bruijn sequences of orders n for every $n \in \mathbb{N}$:*

$$c_{DB} = DB(1)DB(2)DB(3) \dots DB(n) \dots$$

2.2 Cellular Automata

Definition 4 (Cellular automaton). *A* cellular automaton (CA) $\mathcal{A}$ *is a triple* $(Q_\mathcal{A}, r_\mathcal{A}, \delta_\mathcal{A})$ *where* $Q_\mathcal{A}$ *is a finite set of states called the* alphabet, $r_\mathcal{A}$ *is the* radius *of the automaton, and* $\delta_\mathcal{A} : Q_\mathcal{A}^{2r_\mathcal{A}+1} \mapsto Q_\mathcal{A}$ *is the* local rule.

The configurations of a cellular automaton are the configurations over $Q_\mathcal{A}$. *A global behavior is induced and we will note* $\mathcal{A}(c)$ *the image of a configuration* c *given by:* $\forall z \in \mathbb{Z}, \mathcal{A}(c)_z = \delta_\mathcal{A}(c_{z-r}, \dots, c_z, \dots, c_{z+r})$. *Studying the dynamic of* $\mathcal{A}$ *is studying the iterations of a configuration by the map* $\mathcal{A} : Q_\mathcal{A}^\mathbb{Z} \to Q_\mathcal{A}^\mathbb{Z}$.

When there is no ambiguity, we'll note Q, r and δ for $Q_\mathcal{A}$, $r_\mathcal{A}$, $\delta_\mathcal{A}$.

A state $a \in Q_\mathcal{A}$ is said to be *permanent* for a CA $\mathcal{A}$ if for any $u, v \in Q_\mathcal{A}^r$, $\delta(uav) = a$.

2.3 μ-Limit Sets

Definition 5 (Uniform Bernoulli measure). *For an alphabet* Q, *the* uniform Bernoulli measure μ *on configurations over* Q *is defined by:*

$$\forall u \in Q^*, i \in \mathbb{Z}, \mu([u]_i) = \frac{1}{|Q|^{|u|}}.$$

μ will be the only considered measure through this paper, even though these definitions can be generalized for a large set of measures.

For a CA $\mathcal{A} = (Q, r, \delta)$ and $u \in Q^*$, we denote for all $n \in \mathbb{N}$, $\mathcal{A}^n\mu([u]) = \mu\left(\mathcal{A}^{-n}([u])\right)$.

Definition 6 (Persistent set). *For a CA* $\mathcal{A}$, *we define the* persistent set $L_\mu(\mathcal{A}) \subseteq Q^*$ *by:* $\forall u \in Q^*$:

$$u \notin L_\mu(\mathcal{A}) \iff \lim_{n\to\infty} \mathcal{A}^n\mu([u]_0) = 0.$$

Then the μ-limit set *of* $\mathcal{A}$ *is* $\Lambda_\mu(\mathcal{A}) = \left\{c \in Q^\mathbb{Z} : L(c) \subseteq L_\mu(\mathcal{A})\right\}$.

Remark 1. As said in [8], μ-limit sets are closed and shift-invariant. Two μ-limit sets are therefore equal if and only if their languages are equal.

Definition 7 (μ-nilpotency). *A CA* $\mathcal{A}$ *is said to be* μ*-nilpotent if* $\Lambda_\mu(\mathcal{A}) = \{a^\mathbb{Z}\}$ *for some* $a \in Q_\mathcal{A}$ *or equivalently* $L_\mu(\mathcal{A}) = a^*$.

The question of the μ-nilpotency of a cellular automaton is proved undecidable in [2]. The problem is still undecidable with CA of radius 1 and with a permanent state. We will reduce all other properties to this problem.

Remark 2. The set of normal configurations has measure 1 in $Q^\mathbb{Z}$. Which means that a configuration that is randomly generated according to measure μ will be μ-almost surely a normal configuration.

The following lemma translates the belonging to the μ-limit set in terms of density in images of a generic configuration.

Lemma 1. *Given a CA $\mathcal{A}$ and a finite word u, for any generic configuration c:*

$$u \in L_\mu(\mathcal{A}) \Leftrightarrow d_{\mathcal{A}^n(c)}(u) \nrightarrow 0 \text{ when } n \to +\infty$$

Example 1 (MAX). We consider here the "max" automaton $\mathcal{A}_M$: the alphabet contains only two states 0 and 1. The radius is 1 and $\delta_{\mathcal{A}_M}(x, y, z) = \max(x, y, z)$.

The probability to have a 0 at time t is the probability to have 0^{2t+1} on the initial configuration, which tends to 0 when $t \to \infty$ for the uniform Bernoulli measure, so 0 does not appear in the μ-limit set. And finally $\Lambda_\mu(\mathcal{A}_M) = \{^\infty 1^\infty\}$.

The limit set of a cellular automaton is defined as $\Lambda(\mathcal{A}) = \bigcap_{i\in\mathbb{N}} \mathcal{A}^i(Q^{\mathbb{Z}})$, so $\Lambda(\mathcal{A}_M) = (^\infty 10^*1^\infty) \cup (^\infty 0^\infty) \cup (^\infty 10^\infty) \cup (^\infty 01^\infty)$. Actually, we can prove that this limit set is an example of limit set that cannot be a μ-limit set.

2.4 Properties of μ-Limit Sets

Through all this paper, the alphabets of CA will be finite subsets of a countably infinite set $\{\alpha_0, \alpha_1, \alpha_2 \dots\}$. A *property* of μ-limit sets is a family $\mathcal{P}$ of μ-limit sets of CA and any μ-limit set in $\mathcal{P}$ is said to have this property.
A property $\mathcal{P}$ is said to be *nontrivial* if there exist CA $\mathcal{A}_0$ and $\mathcal{A}_1$ such that $\Lambda_\mu(\mathcal{A}_0) \in \mathcal{P}$ and $\Lambda_\mu(\mathcal{A}_1) \notin \mathcal{P}$. For example, μ-nilpotency or the appearance of some state in the μ-limit set are nontrivial properties.

3 Counters

In this section and the following one we describe an automaton $\mathcal{A}_S$, which on normal configurations produces finite segments, separated by a special state #, whose sizes increase with time. First, we will show how to ensure that a normal configuration produces segments, that are designed for computations described in Sect. 5. This means we have to control what is written inside the segments.

We want to erase nearly all of the initial random configuration and only remember some information. This information will be on states $*$ that can appear only at time 0. Each $*$ state sends two counters to its left and right. And these counters will erase everything except a younger counter. Therefore, when two counters meet, they compare their age, and the younger erases the older. If they have the same age, they stop and write a #. The notion of counters was introduced in [3] and used in [1].

Counters are composed of two signals, one faster than the other, and the age of the counter is computed between them. The whole construction (both signals and the binary counter in it) is called a *counter*. Therefore, everything on the initial configuration is erased and forgotten, except for $*$ states. Indeed, even if some counters exist on the initial configuration, they will be older than counters from $*$ when they meet, and hence erased.

For what follows, we will need to have random bits on the #, so we use two states $*_0$ and $*_1$ instead of the unique state $*$. And the bit i is transfered from an $*_i$ to the # produced on its left. We then have $\#_0$ and $\#_1$ instead of #. When it makes no difference, we will speak of # and $*$ for $\#_i$ and $*_i$.

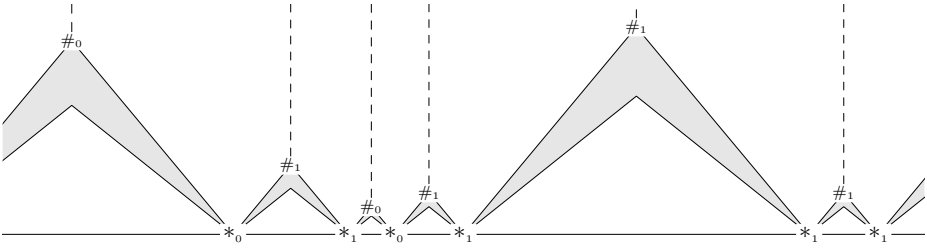


Fig. 1. When two counters launched by a $*$ meet, a # delimiter is produced and counters disappear. Information between $*$ states on the initial configuration is lost when it meets a gray area..

The initialization of a configuration is illustrated in Fig. 1. Counters are schematized by gray areas. # delimiters remain, and we will see later what happens to them.

4 Merging Segments

We saw in Sect. 3, how a special state $*$ on the initial configuration gave birth to counters protecting everything inside them until they meet some other counter born the same way. In this section, we will describe the evolution of the automaton $\mathcal{A}_S$ after this time of initialization. When two counters of the same age meet, they disappear and a # is produced.

Definition 8 (Segment). *A* segment *u is a subword of a configuration delimited by two # and containing no # inside* $\left(u \in \#\left(Q \setminus \{\#\}\right)^* \#\right)$*. The* size *of a segment is the number of cells between both #.*

We have created the segments we were looking for, and as we control what is written inside them, we will be able to perform some computation. This will be described in Sect. 5, we will first see how to let segments grow through time.

When a # is produced in automaton $\mathcal{A}_S$, it sends a signal on its right to detect the first # on its right. If the signal catches the inside of a counter still in activity before reaching a #, it waits until this counter produces a #. Then both # have recognized each other and the segment between them becomes "conscious". It launches a computation inside itself, and this will be the concern of Sect. 5. But as we will need arbitrarily large space for computation, we will remove small segments and replace them by larger ones. Therefore, at some times, we will erase some #, and the segments that were formerly separated will join their space. We describe here mechanisms that lead to this merging process, and then

see how it behaves with μ-limit sets. This happens inside any segment, and in parallel with the computation from Sect. 5.

A segment is said to be *well-formed* if it is delimited by two # that have themselves been created by $*$ states on the initial configuration. To construct $\mathcal{A}_S$, we additionally attribute a color to each segment. There will be Red (R) and Blue (B) segments. So we will have 4 states to replace $*_0$ and $*_1$: $*_{r0}$, $*_{r1}$, $*_{b0}$ and $*_{b1}$. We still use $*$ to refer to any of them indistinctly. An initial segment has color R when produced by $*_{r0}$ or $*_{r1}$, and else color B. The random bit is still transfered to the # on the left as described in Sect. 3.

We require that any segment stores and updates its age since the initial configuration. We'll add two counters (one on each side of the segment) to perform this task. Moreover, we want to ensure that the storage of the age of a well-formed segment does not need more than $\lfloor\sqrt{(n)}\rfloor$ cells where n is the size of the segment. This means we need to know the size of the segment. This can be computed and stored with space $\log(n)$. To maintain the property, segments will merge when the age becomes too large for them.

Suppose we use an alphabet of size $K \geq 3$ to store the age, then the space used in a segment becomes too large at some time K^i with $i \in \mathbb{N}$ (when $i+1 \geq \lfloor\sqrt{(n)}\rfloor$). Every segment has to decide whether it will need to merge, and to tell its neighbors before time K^i. At that time, any segment that needs more space will merge according to the following conditions:

1. if none of its neighbors want to merge, it merges with the left one,
2. if one only of its neighbors wants to merge, it merges with that one,
3. if both its neighbors want to merge, it merges with the left one except if this neighbor has the same color, and the right one has the other color.

Remark 3. Each segment can decide in $(i+1)^2$ steps, if it is larger than $(i+1)^2$. Then it can write on each side if it wants to merge at time K^i or not in less than $2(i+1)^2$ steps. If a segment wants to merge, it can check its neighbors' will on both sides and decide its own behavior in $(i+1)^2$.

The # between two segments that merge together is erased with the age counters around it. Then another cycle starts on the left side of the new segment. Many successive # have possibly disappeared, so the merging is not necessarily two segments becoming one but many segments becoming one. As at least one # has been erased inside the new segment, we use the bit from the leftmost $\#_i$ erased to determine the color of the new segment. If $i = 0$, it will be R, and else B.

We call *initial* segment, a well-formed segment such that only one cell inside it contained a $*$ on the initial configuration. That is, a segment that is well-formed and not created from a merging. And we call *successor* segment, a segment well-formed but not initial, that is, created by a merging of well-formed segments that are its *predecessors*. We so define a set of predecessors at each time.

Remark 4. When two segments merge at time K^i, at least one of them wanted to merge, which means one of them was smaller than $(i+1)^2$.

If three or more segments merge at time K^i, they all wanted to merge, so they were all smaller than $(i+1)^2$.

$\forall i \in \mathbb{N}$, at time $t > K^i$, any segment has a size greater than $(i+1)^2$. This is clear since any segment smaller would have merged at time K^i.

Remark 5. Red and blue segments are initially randomly distributed according to the uniform measure. When some segments merge, the new color is chosen independently from the colors of predecessors or neighbors, and only according to a random bit, so the distribution of colors remains random.

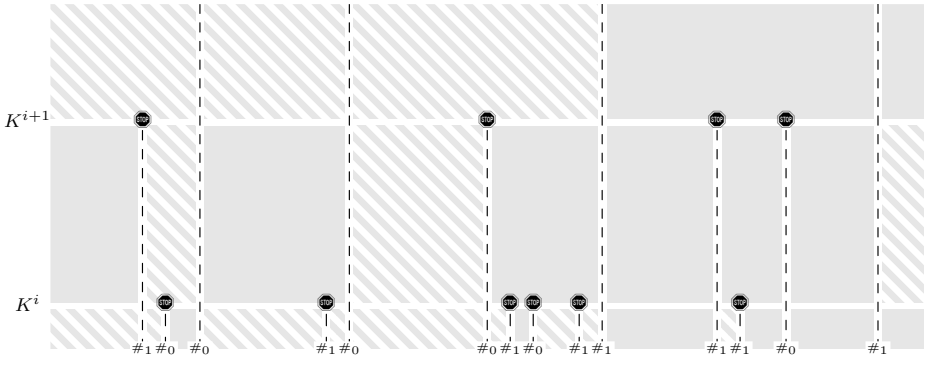


Fig. 2. A # remains until two segments merge. Blue segments are in plain gray, red ones in hashed gray. At time K^i, small segments merge together or with their left neighbor.

The general behavior of the segments among themselves is illustrated in Fig. 2.

Thanks to the following claim, we will be able to prove the first important result for automaton $\mathcal{A}_S$: Prop. 1, which says that # states tend to be sparse enough to be left outside of $\Lambda_\mu(A_S)$.

Claim. The density of cells outside well-formed segments on a normal configuration tends to 0 as time passes.

Proposition 1. *There is no # in the μ-limit set of $\mathcal{A}_S$.*

This comes from the fact that well-formed segments tend to cover the image of a normal configuration. As their growth is permanent, # states are eventually separated by arbitrarily large words.

In Sect. 5, we will need to have an upper bound on the size of a large proportion of segments. We will prove such a bound in the following lemma.

A segment is said to be *acceptable* if it is well-formed and if its size is $n \leq K^{i/4}$ at time $K^i \leq t < K^{i+1}$. In the sequel, we consider the automaton on a normal configuration c_N.

We will show that large segments exist with low probability. First we use the merging protocol, and the colors to justify that a lot of segments rarely merge all together. Then, we show that large initial segments are quite unlikely. In both cases, we give bounds on the probability of large segments.

We have the next lemma:

Lemma 2. *The density of non acceptable segments tends to* 0 *as time passes.*

To prove this, each non acceptable segment is seen as either initial, the product of a merging, or a successor of some non acceptable segment. In the third case, we consider the first predecessor that was in another case and show that it concerns few segments only.

Denote $S_t, t \in \mathbb{N}$ the set of acceptable segments successors of acceptable segments at time t. This set contains all useful segments, it is finite for any time, and it does not depend of the initial configuration.

Remark 6. The same proof as for previous lemma shows that $d_{c_N}(S_t) \to_{t\to\infty} 1$.

Thanks to this, we only need to look at the behavior of the automaton inside segments of S_t, the words that will remain in the μ-limit set will be the words that appear often in these segments.

5 Rice Theorem

The idea here is to copy the principle of the proof of Rice's theorem for limit sets from [7]. We want to reduce any nontrivial property over μ-limit sets to the μ-nilpotency of a CA $\mathcal{H}$ of radius 1 having a permanent state q.

First we construct an automaton to prove the following proposition:

Proposition 2. *For any CA $\mathcal{H}$ of radius* 1*, where a state q is permanent, and CA $\mathcal{A}$, there effectively exists a CA $\mathcal{B}$ such that:*

- *if $\mathcal{H}$ is μ-nilpotent, $\Lambda_\mu(\mathcal{B}) = \Lambda_\mu(\mathcal{A})$.*
- *if $\mathcal{H}$ is not μ-nilpotent, $\Lambda_\mu(\mathcal{B}) = \alpha^{\mathbb{Z}}$, for some chosen $\alpha \in Q_\mathcal{B} \setminus (Q_\mathcal{A} \cup Q_\mathcal{H})$.*

Then, given a property $\mathcal{P}$, for two automata $\mathcal{A}_1$ and $\mathcal{A}_2$, such that one exactly of $\Lambda_\mu(\mathcal{A}_1)$ and $\Lambda_\mu(\mathcal{A}_2)$ has property $\mathcal{P}$, we construct $\mathcal{B}_1$ and $\mathcal{B}_2$. If there existed an algorithm to decide whether a μ-limit set has property $\mathcal{P}$, we could use this algorithm with $\mathcal{B}_1$ and $\mathcal{B}_2$. If only one among $\Lambda_\mu(\mathcal{B}_1)$ and $\Lambda_\mu(\mathcal{B}_2)$ has property $\mathcal{P}$, they have different μ-limit sets and $\mathcal{H}$ is necessarily μ-nilpotent. In the other case, their μ-limit sets cannot be $\Lambda_\mu(\mathcal{A}_1)$ and $\Lambda_\mu(\mathcal{A}_2)$, and $\mathcal{H}$ is not μ-nilpotent. This would be a contradiction with [2], where μ-nilpotency was proved undecidable.

5.1 Construction

The automaton $\mathcal{B}$ is based upon $\mathcal{A}_S$ from previous sections. The whole construction of counters and segments is exactly as described in Sect(s). 3 and 4. We now describe the computation in a segment. We will only concern ourselves with well-formed segments as, for a normal configuration, every cell is eventually reached by such a segment. We partially test the μ-nilpotency of $\mathcal{H}$ in every segment, if the test is successful, we simulate $\mathcal{A}$, meaning we write an image of a prefix of the de Bruijn configuration, if not, we write the uniform word α^n.

Remark 7. We will use the de Bruijn configuration c_{DB} over the alphabet $Q_{\mathcal{A}}$ defined in 2.1. Thanks to [4], a de Bruijn sequence of order k can be computed in space $O(k)$ and time $O(|Q|^k)$. Therefore we can fill a segment of length n with a prefix of c_{DB} in space $O(\log(n))$ and time $O(n)$.

Remark 8. As q is a permanent state in the radius 1 CA $\mathcal{H}$, it behaves like a *wall*, which means no information can travel through a q state. So a simulation of $\mathcal{H}$ over a word $u \in q\left(Q_{\mathcal{H}} \setminus \{q\}\right)^l q$ needs only space l for any l.

In any well-formed segment of size $n \in \mathbb{N}$, the computation of a Turing machine starts on the left of the segment at every time K^i for $i \in \mathbb{N}$.

1. it measures and stores on each side the segment's size.
2. it simulates $\mathcal{H}$ on every $u \in q\left(Q_{\mathcal{H}} \setminus \{q\}\right)^l q$ for $l \leq \frac{1}{2}\left(\log_{|Q_{\mathcal{H}}|}(K^{i/4})\right) - 1$ during $|Q_{\mathcal{H}}|^l$ timesteps. This is how we test the μ-nilpotency of $\mathcal{H}$. If one of the computed images is not q^l, the segment does not simulate $\mathcal{A}$. If all the images are q^l, the segment simulates $\mathcal{A}$.
3. on the left of the segment, it computes $j(i) = \lfloor \log(\log(i)) \rfloor$.
4. if the segment simulates $\mathcal{A}$, the machine computes a prefix of length n of c_{DB}, then computes and writes its $j(i)$-th image by $\mathcal{A}$. If the segment does not simulate $\mathcal{A}$, the head writes α^n over the segment.
5. the machine stops when the whole computation and writing is over or when it has reached time K^{i+1}. At that time, the machine erases itself, leaving what was written.

Remark 9. Each cell in a segment contains a couple:

- a state for computation, storage of the age or the length of the segment,
- a state from $Q_{\mathcal{A}}$.

The first state (computation) is for most cells left blank, then the couple is seen as a state of $Q_{\mathcal{A}}$.

Remark 10. The machine needs only $O(j(i) + \log(n))$ cells to compute. There are $O(\sqrt{(}n))$ additional cells used to count the age on each side, and $O(1)$ cells for signals moving through the segment. And only $O(j(i)n)$ steps are required to perform it.

When a segment is formed by a merging, the data of its cells (on the first layer) is not removed until a new state of $Q_{\mathcal{A}}$ has to be written.

To prove Prop. 2, we will need two lemmas:

Lemma 3. *There exists i_0 such that, for $i \geq i_0$, the computation is finished before $K^{i+1} - 1$ in every segment that is acceptable at time K^i.*

This is easily proved, since the length of acceptable segments is bounded.

And the second lemma, that will let us test the μ-nilpotency of $\mathcal{H}$:

Lemma 4. *If $\mathcal{H}$ is not μ-nilpotent, there exists $l \in \mathbb{N}$ and $u \in q(Q_{\mathcal{H}} \setminus \{q\})^l q$ such that $\mathcal{H}^{|Q_{\mathcal{H}}|^l}(u) \neq q^{l+2}$.*

To prove it, we just use the fact that words which cannot have any permanent state in their antecedents cannot be persistent.

5.2 $\mathcal{H}$ μ-Nilpotent

In this section, we suppose $\mathcal{H}$ is μ-nilpotent and we will show $\Lambda_\mu(\mathcal{B}) \subseteq \Lambda_\mu(\mathcal{A})$.

First, we make sure that the simulation happens everywhere in this case.

Claim. If $\mathcal{H}$ is μ-nilpotent, every well-formed segment simulates $\mathcal{A}$.

Then we prove the following lemma:

Lemma 5. *If $\mathcal{H}$ is μ-nilpotent, then $\Lambda_\mu(\mathcal{B}) \subseteq \Lambda_\mu(\mathcal{A})$*

The proof needs a description of the content of a segment with computation parts, and words computed by the simulation of $\mathcal{A}$.

Remark 11. Thanks to Lemma 3, any $s \in S_t$ with $|s| = n$ at time $K^i \leq t < K^{i+1}$ contains:

- $O(\sqrt{(n)})$ cells for computation, they will not appear in $L_\mu(\mathcal{B})$.
- a subword of $\mathcal{A}^{j(i)}(c_{DB[0..n-1]})$ computed between times K^i and t.
- a concatenation of subwords of $\mathcal{A}^{j(i-1)}(c_{DB[0..n-1]})$ computed before time K^i that were not erased during the merging.

To prove the lemma, we show that any word of $L_\mu(\mathcal{B})$ appears often in large acceptable segments. Then we prove that if a word appears often in an acceptable segment, thanks to the previous remark, it appears often in at least an image of c_{DB} by $\mathcal{A}$. As c_{DB} is generic, we can conclude.

Now we show the second inclusion:

Lemma 6. *If $\mathcal{H}$ is μ-nilpotent, then $\Lambda_\mu(\mathcal{A}) \subseteq \Lambda_\mu(\mathcal{B})$*

To prove this lemma, we consider a word in $L_\mu(\mathcal{A})$. There exists necessarily a sequence of images $(\mathcal{A}^{t_x}(c_{DB}))_{x\in\mathbb{N}}$ in which the density of u does not tend to 0. We take a sequence $(\tau_x)_{x\in\mathbb{N}}$ of times at which $\mathcal{B}^{\tau_x}$ simulates $\mathcal{A}^{t_x}$. Then, thanks to the regularity property of c_{DB}, we can prove that the density of u in the images of a normal configuration by $\mathcal{B}$ does not tend to 0.

5.3 $\mathcal{H}$ Not μ-Nilpotent

In this case, we first make sure that after some time, $\mathcal{A}$ is not simulated in any segment anymore.

Claim. If $\mathcal{H}$ is not μ-nilpotent, there exists i_0 such that, for all $i \geq i_0$, no well-formed segment at time $K^i \leq t < K^{i+1}$ simulates $\mathcal{A}$.

Then we conclude by saying that any letter different from α has a density that tends to 0. Which forces $\alpha^{\mathbb{Z}}$ to be the unique configuration in $L_\mu(\mathcal{B})$.

Lemma 7. *If $\mathcal{H}$ is not μ-nilpotent, then $\alpha^* = L_\mu(\mathcal{B})$.*

This ends the proof of Prop. 2.

5.4 Rice Theorem

First we need to consider two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ over the same alphabet:

Lemma 8. *For any nontrivial property, there exist CA $\mathcal{A}_1$ and $\mathcal{A}_2$ over the same alphabet $Q_{\mathcal{A}}$ such that one among $\Lambda_\mu(\mathcal{A}_1)$ and $\Lambda_\mu(\mathcal{A}_2)$ has this property, and the other not.*

We can prove this lemma by taking multiple copies of each state of both automata, in order to get an alphabet which size is the least common multiple of both sizes.

And finally, we complete the proof of the theorem:

Theorem 1. *Any nontrivial property of μ-limit sets of cellular automata is undecidable.*

As announced, from $\mathcal{A}_1$ and $\mathcal{A}_2$, we construct $\mathcal{B}_1$ and $\mathcal{B}_2$, then deciding a property $\mathcal{P}$ leads to deciding the μ-nilpotency of $\mathcal{H}$.

6 Conclusion

The result presented in this article is that no algorithmic property over μ-limit sets can be decided, except for trivial ones. This, as [1], shows the complexity and hence the interest of this object. We have the same restriction as in [7], that is, we work on an unlimited set of states. One property at least becomes decidable if we limit the set of possible states, it is having the fullshift as μ-limit set. Which is equivalent to being surjective.

In [5], it was proved that surjectivity was the only decidable problem on limit sets with a fixed alphabet. This extension could perhaps be adapted to μ-limit sets and then show another parallel between limit and μ-limit sets.

This is also another use of counters and segments, showing how powerful this tool can be for cellular automata. Especially concerning μ-limit sets.

References

1. Boyer, L., Delacourt, M., Sablik, M.: Construction of μ-limit sets. In: Kari, J. (ed.) JAC 2010, vol. 13, pp. 76–87. TUCS Lecture notes, Turku (2010)
2. Boyer, L., Poupet, V., Theyssier, G.: On the complexity of limit sets of cellular automata associated with probability measures. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 190–201. Springer, Heidelberg (2006)
3. Delacourt, M., Poupet, V., Sablik, M., Theyssier, G.: Directional dynamics along arbitrary curves in cellular automata. To be Pubished in Theoretical Computer Science (2011)
4. Fredricksen, H., Kessler, I.: Lexicographic compositions and debruijn sequences. Journal of Combinatorial Theory, Series A 22(1), 17–30 (1977)
5. Guillon, P., Richard, G.: Revisiting the rice theorem of cellular automata. In: Marion, J.Y., Schwentick, T. (eds.) STACS 2010, vol. 5, pp. 441–452. LIPIcs, Schloss Dagstuhl (2010)

6. Hurley, M.: Ergodic aspects of cellular automata. Ergodic Theory and Dynamical Systems 10, 671–685 (1990)
7. Kari, J.: Rice's theorem for the limit sets of cellular automata. Theoretical Computer Science 127(2), 229–254 (1994)
8. Kůrka, P., Maass, A.: Limit sets of cellular automata associated to probability measures. Journal of Statistical Physics 100, 1031–1047 (2000)
9. von Neumann, J.: Theory of Self-Reproducing Automata. University of Illinois Press, Champaign (1966)
10. Rice, H.G.: Classes of recursively enumerable sets and their decision problems. Transactions of the American Mathematical Society 74(2), 358–366 (1953)

Fault-Tolerant Compact Routing Schemes for General Graphs

Shiri Chechik

Department of Computer Science and Applied Mathematics,
The Weizmann Institute of Science, Rehovot, Israel
shiri.chechik@weizmann.ac.il

Abstract. This paper considers compact fault-tolerant routing schemes for weighted general graphs, namely, routing schemes that avoid a set of failed (or *forbidden*) edges. We present a compact routing scheme capable of handling multiple edge failures. Assume a source node s contains a message M designated to a destination target t and assume a set F of edges crashes (unknown to s). Our scheme routes the message to t (provided that s and t are still connected in $G \setminus F$) over a path whose length is proportional to the distance between s and t in $G \setminus F$, to $|F|^3$ and to some poly-log factor. The routing table required at a node v is of size proportional to the degree of v in G and some poly-log factor. This improves on the previously known fault-tolerant compact routing scheme for general graphs, which was capable of overcoming at most 2 edge failures.

1 Introduction

Routing is one of the most fundamental problems in distributed networks. A routing scheme is a mechanism that allows delivering a message from a source node s to a target node t. In the routing process, each node in the network may receive packets of information and has to decide if the message already reached its destination or not. If the message did not reach its destination then the node has to forward the message to one of its neighbors, using only its local information and the header of the message, containing the label of the destination and possibly some other data. A key concern in designing a routing scheme is to minimize the worst case multiplicative stretch, namely, the maximum ratio between the length of a path obtained by the routing scheme and the length of the shortest path between the source and the destination. Another important goal is to minimize the size of the routing tables stored at the nodes. Subsequently, the focus in designing a routing scheme is often on the tradeoff between the size of the routing tables and the maximum stretch of the resulting routes.

The problem of designing compact and low stretch routing schemes has been extensively studied (e.g. [17,3,5,16,9,12,15]). The first tradeoff between the size of the routing tables and the maximum multiplicative stretch of the routing scheme was considered by Peleg and Upfal [17]. In this paper, the total size of the routing tables was considered (as opposed to the maximum table size of the nodes) and

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 101–112, 2011.

only unweighted graphs were considered. Later, weighted graphs were considered by Awerbuch et al. [3] and a bound on the maximum table size was achieved. The weights on the edges correspond to distances, namely, the distance of a path is the sum of the weights of its edges. Awerbuch et al. show how to construct a routing scheme with maximum table size of $\tilde{O}(n^{1/k})$ and with multiplicative stretch dependent on k ($O(k^2 9^k)$) for any integer $k \geq 1$. Further improvements were later introduced by Awerbuch and Peleg [5] for general integer $k > 1$, by Cowen [9] for $k = 3$ and by Eilam et al. [12] for $k = 5$. These tradeoffs were later improved by Thorup and Zwick [18], obtaining the best one known so far. They present a routing scheme with table size of $\tilde{O}(n^{1/k})$ and a multiplicative stretch of $2k-1$ using handshaking by which the source and target agree on an $o(\log^2 n)$ bit header that is attached to all packets, or a stretch of $4k-5$ without using handshaking. Corresponding lower bounds were presented in [17,13,14,19].

In this paper, we consider a natural and significant extension of routing schemes for general weighted graphs, that may better suit many network settings. Suppose that some of the links crash from time to time and it is still required to deliver messages between the nodes, if possible, without recomputing the routing tables and the labels. Given a set F of edge failures, the multiplicative stretch is with respect to the distance between the source node and the destination node in the surviving graph $G \setminus F$. The objective is once again to minimize both the routing table sizes and the stretch. This extension was suggested in [8,20], which consider this problem for graphs of bounded treewidth or cliquewidth. It is shown how to assign each node a label of size $O(\log^2 n)$ (with some dependency on the tree/clique width) such that given the labels of the source and target and the labels of a set F of "forbidden" vertices, the scheme can route from the source to the target on the shortest path in $G \setminus F$.

Later, in [1], the same extension was considered for unweighted graphs of bounded doubling dimension. It is shown how to construct a labeling scheme for a given unweighted graph of doubling dimension α such that for any desired precision parameter $\epsilon > 0$, the labeling scheme stores an $O(1+\epsilon^{-1})^{2\alpha} \log^2 n$-bit label at each vertex. Given the labels of two end-vertices s and t, and the labels of a set F of "forbidden" vertices, the scheme can route on a path of length at most $1+\epsilon$ times the distance from s to t in $G \setminus F$.

Note that in both [8] and [1] the assumption is that the labels of the faulty nodes are known to the source.

For weighted general graphs, the design of fault-tolerant compact routing schemes was considered in [6]. However, the scheme of [6] only dealt with up to 2 edge failures and bounded only the total size of the routing tables at all the nodes. It is shown how to construct a routing scheme for a given parameter k, that in the presence of a forbidden edge set F of size at most 2 (unknown to the source), routes the message from the source s to the destination t over a path of length $O(k \cdot \mathbf{dist}(s,t,G \setminus F))$, where $\mathbf{dist}(s,t,G \setminus F)$ is the distance from s to t in $G \setminus F$. The total amount of information stored in the vertices of G is $O(kn^{1+1/k} \log(nW) \log n)$, where W is the weight of the heaviest edge in the graph (assuming the minimal weight is 1).

Our contributions In this paper we present a compact fault-tolerant routing scheme for weighted undirected general graphs. We manage to design a compact routing scheme that handles multiple edge failures. More specifically, we prove the following theorem.[1]

Theorem 1. *Given a graph $G = (V, E)$ with edge weights ω such that $\omega(e) \in [1, W]$ for every edge e and a parameter k, one can efficiently construct a routing scheme that given a source node s and a target node t, in the presence of a set of failures F (unknown to s), can route a message from s to t in a distributed manner over a path of length at most $O(|F|^2 \cdot (|F| + \log^2 n) \cdot k \cdot \mathbf{dist}(s, t, G \setminus F))$. The scheme requires assigning to each node a label of length $O(\lceil \log(nW) \rceil \cdot \log n)$ bits and the routing table of a node v is of size at most $O(\lceil \log(nW) \rceil \cdot k \cdot n^{1/k} \cdot \deg(v) \cdot \log^2 n)$ bits. The message passed during the routing process is of size $O(|F| \cdot \log n)$ bits.*

2 General Framework

In this section we outline the general structure of our routing scheme. Let $G(V, E)$ be an n-node undirected weighted graph with edge weights ω such that $\omega(e) \in [1, W]$ for every edge e (hence nW is an upper bound on the diameter). For a given graph H and a tree T such that $V(T) \subseteq V(H)$, let $H|_T$ be the subgraph of H induced on the vertices of T.

Our results are based on the well known construction of tree cover, defined as follows.

Tree covers: Let $G(V, E)$ be an undirected graph with edge weights ω, and let ρ, k be two integers. Let $B_\rho(v) = \{u \in V \mid \mathbf{dist}(u, v, G) \leq \rho\}$ be the ball of vertices of (weighted) distance at most ρ from v. A *tree cover* $\mathbf{TC}(G, \omega, \rho, k)$ is a collection $\mathcal{T} = \{T_1, \ldots, T_\ell\}$ of rooted trees in G, with $V(T) \subseteq V$ and a root $r(T)$ for every $T \in \mathcal{T}$, with the following properties:

(i) For every $v \in V$ there exists a tree $T \in \mathcal{T}$ such that $B_\rho(v) \subseteq T$.
(ii) For every $T \in \mathcal{T}$ and every $v \in T$, $\mathbf{dist}(v, r(T), T) \leq (2k-1) \cdot \rho$.
(iii) For every $v \in V$, the number of trees in $\mathcal{T}$ that contain v is $O(k \cdot n^{1/k})$.

Proposition 1 ([4,7,16]). *For any ρ and k, there exists a tree cover $\mathbf{TC}(G, \omega, \rho, k)$ constructible in time $\tilde{O}(mn^{1/k})$.*

A basic building block of our routing scheme is a procedure for routing on subtrees of the graph. For that purpose we use the routing scheme on trees of Thorup and Zwick [18]. That scheme uses $(1+o(1)) \log_2 n$-bit label size and no additional information is stored in the nodes.

Let us start with a high level overview of the way our routing scheme operates. Consider vertices $s, t \in V$ and suppose that a message is to be routed from s to t. Our routing process involves at most $\lceil \log(nW) \rceil$ iterations, where iteration i is

[1] Notice that by setting $k = \log n$, the term $n^{1/k}$ in the theorem becomes a constant.

expected to succeed in passing the message in the case where $2^{i-1} < \mathbf{dist}(s,t,G \setminus F) \leq 2^i$. As iteration i handles the possibility that $\mathbf{dist}(s,t,G \setminus F)$ is at most 2^i, it may ignore edges of weight greater than 2^i. Formally, let H_i be the set of G edges of weight greater than 2^i and let G_i be $G \setminus H_i$. Clearly, any two vertices that are connected in G by a path of length at most 2^i are still connected in G_i by the same path. Hence the routing process may be restricted to G_i. To facilitate each iteration i, in the preprocessing phase construct a tree cover $\mathbf{TC}_i = \mathbf{TC}(G_i, \omega, 2^i, k)$. Now for each tree $T \in \mathbf{TC}_i$ invoke our scheme for routing on trees with failures presented in Section 3 on the tree T and the graph G_i to assign each node $v \in T$ a label $L(v,T)$ and a routing table $A_v(T)$.

Each node v stores a routing table A_v, containing all the routing tables $A_v(T)$ together with some unique identifier $id(T)$ for each tree T such that $v \in T$ and $T \in \mathbf{TC}_i$ for some $1 \leq i \leq \lceil \log(nW) \rceil$.

In addition, for each node $t \in V$, let $T_i(t) \in \mathbf{TC}_i$ be the tree containing $B_{2^i}(t)$. The label $\mathcal{L}(t)$ of each node $t \in V$ has to store enough information about each $T_i(t)$ in order to allow routing on the tree $T_i(t)$ to the target t. Specifically, the label $\mathcal{L}(t)$ stores $L(t,T)$ together with the unique identifier $id(T)$ for the tree $T = T_i(t)$, for every $1 \leq i \leq \lceil \log(nW) \rceil$.

The routing process is schematically done as follows. Let F denote the set of failed edges at a given moment. In each iteration i from 1 to $\lceil \log(nW) \rceil$, an attempt is made to route the message from the source s to the target t in the graph $G_i|_{T_i(t)} \setminus F$ using the tree $T_i(t)$ augmented with some additional information to be specified later on. If the routing is unsuccessful, i.e., it is not possible to route to t in $G_i|_{T_i(t)} \setminus F$, then s is informed by the routing scheme that it must proceed to the next iteration. Note that in order to route from s to t in the tree $T_i(t)$, the node s has to be familiar with the label $L(t,T)$ given to t by our scheme for the tree $T = T_i(t)$; this information can be extracted from t's label $\mathcal{L}(t)$.

In order to complete the description of our routing scheme, it remains to present our routing process from s to t in the graph $G_i|_{T_i(t)} \setminus F$. In what follows, we focus on describing our routing scheme in $H|_T \setminus F$ for a given graph H and a tree T, and describe the information stored in both the preprocessing phase and the routing phase.

3 Routing on a Tree with Faults

In this section we consider a given graph H and a tree T in H, and design a routing scheme such that when a set of edges F fails, if s and t are still connected in $H|_T \setminus F$, then our routing procedure manages to deliver a message from s to t on a path of length proportional to the depth of T, to F^3 and to some poly-log factor. More specifically, we prove the following lemma.

Lemma 1. *Consider a graph H with maximum edge weight W_H and a tree T of H ($E(T) \subseteq E(H)$). There is an efficiently constructible routing scheme such that given a source node s and a target node t, in the presence of a set of failures F (unknown to s), if s and t are connected in $H|_T \setminus F$, will deliver a message*

from s to t on a path of length $O(f^2(f \cdot diam(T) + f \cdot W_H + \log^2 n \cdot diam(T)))$, where $f = |F|$ and $diam(T)$ is the diameter of T. The size of the routing labels used by the scheme is $O(\log n)$ bits and the routing table stored at a node v is of size $O(\log^2 n \cdot \deg_H(v))$ bits, where $\deg_H(v)$ is the degree of v in the graph H.

The routing scheme described in this section is strongly based on the result of Duan and Pettie [11] on connectivity oracles with edge failures. We first review that result, and later show how to implement it in a distributed setting to achieve our fault-tolerant routing scheme. Their algorithm operates on a given spanning tree T of G. The algorithm traverses the tree T according to some Euler tour and constructs the list $L(T)$ of the vertices of $V(G)$ in the order in which they were encountered during this Euler tour, keeping only the first occurrence of each vertex. For every v, let $\ell(v)$ denote v's index in $L(T)$. The algorithm then considers the adjacency matrix M of G, according to the ordering $L(T)$, and constructs a *range reporting* data structure on M (concretely, using the range reporting data structure of Alstrup et al. [2]). Duan and Pettie [11] observe that removing f edges from T partitions it into $f+1$ connected subtrees and splits $L(T)$ into at most $2f+1$ intervals, where the vertices of each connected subtree are the union of some subset of these intervals. Hence in order to decide connectivity in $G \setminus F$, it is enough to determine, for all pairs of the $2f+1$ intervals, if there is an edge in $E \setminus F$ connecting them. This can be done efficiently by the range reporting data-structure constructed on M.

For the purpose of implementing the range reporting data structure in a distributed manner, we use a less efficient range reporting data structure. We first describe the well-known centralized range reporting data structure that is used in our construction, and later, in Subsections 3.1 and 3.2, we show how to implement this data structure in a distributed setting (which may be of independent interest).

Consider a boolean matrix $\tilde{M}$, in which queries of the following form need to be answered: "Given a range of rows (r_1, r_2) and a range of columns (c_1, c_2), return all cells that contain 1 in the matrix $\tilde{M}$ and are in the range of rows (r_1, r_2) and the range of columns (c_1, c_2)." The problem can be transformed to the following *planar range reporting* problem. Given a set P of N points in the plane, construct a data structure that can answer queries of the form: "return all points in the rectangle $[x_1, x_2] \times [y_1, y_2]$". This can be done by creating a set of points $P(\tilde{M})$ containing a point $p(c)$ for each cell c in the matrix $\tilde{M}$ that contains 1 and setting its x-coordinate to be the row of the cell c and its y-coordinate to be the column of c. Now a query of the form $(r_1, r_2) \times (c_1, c_2)$ on $\tilde{M}$ can be answered by finding all points $P(\tilde{M})$ inside the rectangle $[r_1, r_2] \times [c_1, c_2]$. A data structure that can answer efficiently planar range reporting queries can be constructed as follows (cf. [10]). Given a set P of N nodes in the plane, first construct a main balanced binary tree $\mathcal{M}$ built on the x-coordinate of the points in P. For any internal or leaf node v in $\mathcal{M}$, let $P(v)$ be the subset of points of P corresponding to the subtree of v in $\mathcal{M}$. For any internal or leaf node v in $\mathcal{M}$, store a balanced binary tree $T(v)$ built on the y-coordinate of the points in $P(v)$. In addition, form a connected list chaining the leaves of $T(v)$, namely, provide

each leaf u with a pointer to the next leaf with the lowest y-coordinate that is greater equal to the y-coordinate of u. Now given a query $[x_1, x_2] \times [y_1, y_2]$, the query algorithm first selects $O(\log n)$ canonical subsets that together contain all the points whose x-coordinate lie in $[x_1, x_2]$. Each such subset corresponds to a node in $\mathcal{M}$. In each such internal node $v \in \mathcal{M}$, the query algorithm searches for all nodes whose y-coordinate lie in $[y_1, y_2]$ using the search tree $T(v)$. The search on $T(v)$ takes $O(\log n + k')$ time, where k' is the number of nodes that lie in $[y_1, y_2]$ in $T(v)$. All in all, the query time is $O(\log^2 n + k)$, where k is the number of nodes reported. Note that if we want to report only $k'' \leq k$ points, then the query can be answered in $O(\log^2 n + k'')$ time.

Next, we turn to describe how to implement this range reporting data structure on M in a distributed setting. In Subsection 3.1 we describe the preprocessing phase, namely, the data that needs to be stored at the nodes in the preprocessing phase, Subsection 3.2 describes the routing phase and finally in Subsection 3.3 we prove the correctness of Lemma 1.

3.1 Preprocessing

In the preprocessing phase, construct the main balanced binary tree $\mathcal{M}$ built on the nodes V according to their order in $L(T)$, i.e., their indices $\ell(v)$. For an internal node $\tilde{v} \in \mathcal{M}$, let $P(\tilde{v})$ be the set of nodes in V corresponding to the subtree of $\tilde{v}$ in $\mathcal{M}$. For a set of nodes S, denote by $E_{out}(S)$ the set of edges in $E \setminus E(T)$ with exactly one endpoint in S. For an edge $e = (u, v) \in E_{out}(S)$, where w.l.o.g $u \in S$ and $v \notin S$, denote the incident node to e that is not in S by $Out(e, S) = v$. Let $L(S)$ be the set of edges $E_{out}(S)$ sorted by $Out(e, S)$ according to the order $L(T)$. For each internal node $\tilde{v} \in \mathcal{M}$, construct a balanced binary tree $T(\tilde{v})$ on $E_{out}(P(\tilde{v}))$ according to the order $L(P(\tilde{v}))$, namely, $T(\tilde{v})$ is a balanced binary tree whose nodes set is the set of edges $E_{out}(P(\tilde{v}))$.

Our algorithm looks for ways of progressing from its currently familiar "piece" of the network represented as an interval $I = (v_i, ..., v_j)$ of the ordered list $L(T)$ as explained earlier, to the "piece" containing the destination, which is another such interval I', disconnected from I by the faults of F. Note that in order to find all edges connecting some interval $I = (v_i, ..., v_j)$ with some other interval I', we need to check a set $X(I, I')$ of $O(\log n)$ internal nodes in $\mathcal{M}$. Consider some node $u \in V$, and let p be the path from the leaf representing u in $\mathcal{M}$ to the root of $\mathcal{M}$, each non-leaf node x on the path p has two children, one that is part of the path p and another node y, if y is a left child, add it to $S_L(u)$, otherwise to $S_R(u)$. The node u itself is added to both $S_L(u)$ and $S_R(u)$. Note that the set $X(I, I')$ is a subset of $S_R(v_i) \cup S_L(v_j)$. Every node u stores an identifier of the internal nodes in $S_R(u) \cup S_L(u)$. As explained above, for each node $\tilde{v}$ in $S_R(u) \cup S_L(u)$, a balanced search tree $T(\tilde{v})$ is constructed, and the identifier of $\tilde{v}$ will contain an indication of the edge represented by the root of $T(\tilde{v})$. In addition, each internal node w in $T(\tilde{v})$ represents an edge $e = (x, y)$, where one node, say x, is inside $P(v)$ and the other, y, is outside it. The routing table of x stores the edges of the left and right children of w in $T(\tilde{v})$ as well as the ranges they represent.

To summarize, the routing table $A_v(T)$ contains the identifiers of the internal nodes $S_R(v) \cup S_L(v)$, where the identifier of each internal node $\tilde{u} \in S_R(v) \cup S_L(v)$ contains the edge represented by the root of $T(\tilde{u})$. In addition, for every internal node $\tilde{u} \in \mathcal{M}$ such that $v \in P(\tilde{u})$ and every edge $(v, y) \in E_{out}(P(\tilde{u}))$, let w be the internal node in $T(\tilde{u})$ that represents (v, y), the routing table $A_v(T)$ stores the edges of the left and right children of w in $T(\tilde{u})$ as well as the ranges they represent (together with some identifier of the tree $T(\tilde{u})$).

For a node $v \in V$, the label $L(v, T)$ is the concatenation of the label given to v by the routing scheme of [18] on the tree T and the index $\ell(v)$.

The routing table $A_v(T)$ of v also stores the label $L(v, T)$.

3.2 The Routing Process

In this section we show how to route a message from a source s to a target t on a single tree T with faults, allowing it to bypass the failures. After the failures of at most f edges, the tree is divided into at most $2f + 1$ intervals, and the goal is to reconnect these intervals, if possible. In the beginning of the routing process, the failed edges are not known to the nodes (except for their endpoints), so clearly, the different intervals are not known, and of course the way to reconnect the intervals is not known. During the routing process, some of this data is revealed and is attached to the header of the message. More specifically, the message accumulates information on the set of known intervals and the discovered edges that reconnect these intervals. Let $\mathcal{I}$ be the graph obtained by representing each discovered interval as a node and connecting two nodes if the process has already found an edge connecting the intervals they represent. The message carries along with it a copy of the graph $\mathcal{I}$. The goal is to reach as many intervals as possible in order to eventually reach t. As the nodes do not necessarily know all the failures, some of the intervals are "fake", in the sense that inside an interval there may be a failure (which the message still does not know about) that splits this interval, possibly into 3 intervals, where two of them are already connected. The intuition is that as long as the message does not encounter this failure, we do not care that the data it stores is imprecise, namely, if the message never encounters this failure, it means that this failure does not disturb our routing process and therefore we do not care about it. The other possibility is that eventually the message will encounter this failure, which will force it to update the set of intervals it carries along. Notice that the latter can happen at most f times, where f is the number of failed edges.

We now describe the routing process more formally. For simplicity, we first describe a solution where the message forwarded during the routing process is of size $O(|F|^2 \cdot \log n)$ bits, we later describe the small modifications needed to reduce the message size to $O(|F| \cdot \log n)$ bits. The header of the message contains the following additional data: it stores the set of currently known intervals, and for each pair of intervals I, I' it stores one of the following three items: a discovered edge $e(I, I')$ connecting them, an indication $discon(I, I')$ that these two intervals can not be connected, or an indication $Unknown(I, I')$ that it is still unknown if these two intervals can be connected. (Notice that some of the intervals are

already connected by the original tree T.) The goal is to explore the pairs of intervals that are not decided yet, in order to eventually reach t if possible, or to conclude that t is not reachable from s in $H|_T \setminus F$.

At the beginning of this process, the source s is unaware of the failures, hence the set of intervals contains only one interval (which represents the entire tree T), and it just tries to route on T as if no failures occurred. The simplest scenario is that the path connecting s and t in T is free from failures and the message arrives at its destination. The more interesting case is when the routing process encounters some failure along the way and thus can not complete the normal routing process successfully.

In case a new failure is detected, the set of intervals is updated accordingly, and the set of *recovery edges* is updated as well, where by a recovery edge we mean an edge that reconnects two intervals that are not connected in $T \setminus F$. Namely, assume the interval I is now split into at most three intervals I_1, I_2, I_3, and consider an edge $e(I, I')$ that was previously believed to connect I to some other interval I'. This edge now connects one of the three intervals I_1, I_2, I_3 to I', and as we know the incident nodes of the edge, there is no problem to identify the right one, say, I_1, and update the information in the message header to include $e(I_1, I')$. In addition, if it is already known that there is no way to reconnect I to some other interval I' (i.e., the message header contains $discon(I, I')$), then it is also impossible to reconnect I' to any of I_1, I_2, I_3 and we update the data in the message header by storing in it the indicators $discon(I_1, I')$, $discon(I_2, I')$ and $discon(I_3, I')$.

At any stage during the routing process, the routing process may be in one of three states. The first state is that the nodes can check if using the recovery edges collected so far it is possible to reach t (for example, by running a DFS on the graph $\mathcal{I}$). In the second state, the nodes can verify if it is impossible to reach t in $H|_T \setminus F$. This case happens when it is already known that all the intervals that are reachable from s can not be connected to any other interval and t is still not reachable from s, where we say that an interval I is reachable from some node $v \in T$ if the interval containing v and the interval I are connected in $\mathcal{I}$. The third state is where it is still unclear if it is possible to reach t in $H|_T \setminus F$.

In the first state, the message is sent to t (assuming no new failures are detected during the remaining part of the journey). In the second state, an indication that t is not reachable in $H|_T \setminus F$ is sent to s. In the third state, pick a pair of intervals (I, I') that have not been decided yet (i.e., such that the message header contains $Unknown(I, I')$) and such that I is reachable from the current node (and therefore also from s). If no such pair exists, it can be determined that the routing is not possible. Next, search for a recovery edge connecting I and I' as follows. First, forward the message to some node in I. Once reaching I, check the potential search balanced trees, namely, the search trees for the interval I (this data can be extracted from the routing tables of the first and last nodes v_i and v_j in the interval I, recall that the set the potential search trees is a subset of $S_R(v_i) \cup S_L(v_j)$). Now for each potential search tree T_1 on an interval $I_1 \subseteq I$, search for a recovery edge reconnecting to I'. The search

on T_1 can be done as follows. First reach the node z containing the root of T_1 (recall that the root of T_1 represents an edge (x, y) where x is in I_1 and y is outside, so $z = x$ and that the edge (x, y) is part of the identifier of T_1). The node z stores in its routing table the left and right children of the root of T_1 as well as the ranges they represent, so the node z knows if it supposed to continue the search on the left or right child. This process continues until reaching the leaves. If the leaf is not in the range I', then it's not hard to see that I_1 and I' can not be connected. Otherwise, there are two subcases. If the edge in the leaf is not faulty, then we found a recovery edge reconnecting I and I'. The second subcase is that this edge is faulty. Recall that each leaf stores an identifier to the next leaf, so we can just check the next leaf, and so on. This is repeated for all $O(\log n)$ potential subintervals of I, until either finding the desired recovery edge connecting I and I' or deciding that it is impossible to connect I to I'.

3.3 Analysis

This section is devoted to proving Lemma 1.

Correctness. We need to prove that if s and t are connected in $H|_T \setminus F$, then t will get the message. This follows almost trivially from [11] and is proved in the following lemma.

Lemma 2. *If s and t are connected in $H|_T \setminus F$, then the message will reach its destination using our routing scheme.*

Proof: Assume s and t are connected in $H|_T \setminus F$. Consider only final intervals, namely, intervals that were not split any more during the routing process. Let $\mathcal{I}_{fin}$ be the final graph $\mathcal{I}$. It's not hard to verify that there must be a path of final intervals in $\mathcal{I}_{fin}$ connecting the final interval of s with the final interval of t. As our algorithm checks all possible pairs of intervals, it will eventually find that path (or some other path p reaching t). Therefore, either the algorithm manages to reach t on p, or it detects another failure on p, but this is in contradiction with the fact that we consider final intervals. ∎

Stretch Analysis.

Lemma 3. *If s and t are connected in $H|_T \setminus F$, then the length of the path obtained by our routing scheme on $H|_T \setminus F$ is at most $O(f^2(f \cdot diam(T) + f \cdot W_H + \log^2 n \cdot diam(T)))$.*

Proof: As mentioned above, there are at most $O(f)$ real intervals. Therefore, at most $O(f^2)$ pairs of real intervals need to be examined. In the beginning, the routing process is unaware of these intervals, so it might check "fake" intervals. At first glance, this appears to be a waste and it seems that we might need to check more than $O(f^2)$ pairs of intervals. But a more careful inspection reveals that it is enough to check only $O(f^2)$ pairs of intervals. To see why this is true, assume we have already checked the pair of intervals I and I' and at some point

discover that I splits into two or three intervals, I_1, I_2, I_3. There are two cases. The first is that we discovered that I and I' can not be connected. In that case we can determine that all I_1, I_2, I_3 can not be connected to I', and actually this works to our advantage, as in one check we saved two or three checks. The second case is where I and I' were connected by an edge $e = (u, v)$ such that $u \in I$ and $v \in I'$. Notice that u must belongs to one of I_1, I_2, I_3. Assume w.l.o.g. that it belongs to I_1. Then we can determine that I_1 and I' are connected by the edge e. In addition, each time we try to discover if two intervals are connected or not, we either determine if this pair can be connected or not, or we discover another failure. As mentioned earlier the latter can happen at most f times. To conclude, there are at most $O(f^2)$ such checks.

Next we bound the maximum length of the path obtained by a single check of two intervals. Assume we want to check the pair of intervals I and I'. It must be the case that one of the intervals is reachable by the edges of T together with the recovery edges discovered so far. Therefore, the path leading to the relevant interval is of length at most $(f+1) \cdot diam(T) + f \cdot W_H$, where $diam(T)$ is the diameter of T. To see this, note that the diameter of each of the connected subtrees of $T \setminus F$ is at most $diam(T)$, and in addition, the path uses at most f recovery edges of weight at most W_H each. Next, we bound the length of the subpath used for checking the pair of intervals I and $I' = [a, b]$ once we are already in I. It's enough to bound the maximum length assuming no new faulty edge was encountered during the way. We know the interval I, and in particular we know its first and last nodes and thus can easily find the id's of the $O(\log n)$ search trees that need to be examined. We now bound the length of one of these search trees T_1. The id of the tree T_1 contains the root node of the tree T_1 and each intermediate node contains the id's of its left and right children and the ranges they represent. Moreover, we assume that the interval is connected, so every two nodes in the interval are reachable using only the edges of T. We start with the root of T_1 and search for the edge e of minimum $\ell(Out(e, S))$ that is equal to or greater than a, where the set S is the set of nodes $P(r(T_1))$ where $r(T_1)$ is the root of T_1. Checking the ranges of children of the root, we know if we need to move left or right, and we continue in this way until reaching the relevant leaf. Note that a child and a parent in the tree T_1 are not necessarily adjacent in T, but their distance is at most $diam(T)$. The depth of T_1 is at most $\log n$, and for each step we might need to travel a distance at most $diam(T)$. Therefore the total length for reaching the relevant leaf for a single search tree is at most $\log n \cdot diam(T)$. Summing over all search trees, the total length is $O(\log^2 n \cdot diam(T))$. Note that once reaching a leaf in one of the search trees, this leaf might represent a faulty edge, therefore we might need to check the next leaf and so on, until either reaching a leaf that is not in I' or finding a non-faulty edge in I'. Notice that this can happen at most f times for all search trees (since the sets of edges of the search trees are disjoint), and in addition, moving from one leaf to the next can be done along a path of length at most $diam(T)$. All in all, the total length of the path traversed on T is $O(f^2(f \cdot diam(T) + f \cdot W_H + \log^2 n \cdot diam(T)))$. ∎

To see why the label sizes are $O(\log n)$, note that the label $L(v,T)$ of a node v is just the concatenation of the label given to v by the routing scheme of [18] on T and the index of v in $L(T)$. In addition, v participated in $O(\log n)$ search trees, and for each such search tree, $O(\log n \cdot \deg_H(v))$ bits are stored in the routing table $A_v(T)$ of v. We get that the routing table $A_v(T)$ is of size $O(\log^2 n \cdot \deg_H(v))$ bits.

Together with Lemmas 2 and 3, Lemma 1 follows.

4 Analysis for the Entire Routing Scheme

In this section we analyze our routing scheme. Let p be a shortest path connecting s and t in $G \setminus F$, and let i be the index such that $2^{i-1} \leq \mathbf{dist}(P) = \mathbf{dist}(s,t,G \setminus F) \leq 2^i$.

The following lemmas prove Theorem 1. Due to space limitations, some of the proofs are deferred to the full paper.

Lemma 4. *The nodes s and t are connected in $G_i|_{T_i(t)} \setminus F$.*

Proof: First note that all edges in p are of weight at most 2^i and thus exist in G_i, and moreover as p is a path in $G \setminus F$, all these edges are non-faulty and thus occur also in $G_i \setminus F$. Moreover, $T_i(t)$ contains all nodes at distance at most 2^i from t. We get that all nodes in p are in $T_i(t)$. The lemma follows. ∎

Lemma 5. *The path obtained by our routing scheme is of length at most $O(f^2 \cdot k \cdot \mathbf{dist}(s,t,G \setminus F)(f + \log^2 n))$.*

Lemma 6. *The size of the routing table of a node v is at most $\lceil \log(nW) \rceil \cdot k \cdot n^{1/k} \cdot \deg(v) \cdot \log^2 n$ bits.*

Lemma 7. *The labels are of size $\lceil \log(nW) \rceil \cdot \log n$ bits.*

Lemma 8. *The message forwarded during the routing process is of size $O(|F|^2 \cdot \log n)$ bits.*

In fact, the message size can be reduced to $O(|F| \cdot \log n)$ bits. To see this, note that in order to decide connectivity in $G_i|_{T_i(t)}$, there is no need to store information on all pairs of intervals, rather it's enough to store at most $|F|$ recovery edges and an indication which pairs of intervals have already been examined. This can be done using $O(|F| \cdot \log n)$ bits (For further discussion of this point see the full paper.).

We note that if we care only for the total size of the routing tables at all the nodes, then it's possible to reduce the stretch bound by a logarithmic factor. This can be achieved as follows. Recall that the algorithm stores for each internal node v in $\mathcal{M}$ a balanced search tree $T(v)$ in a distributed manner. Let I be the interval of nodes corresponds to the internal node v. Instead of storing the tree $T(v)$ distributively on all nodes in I, the algorithm can just pick an arbitrary node x in I and store the entire tree $T(v)$ in x. It's not hard to see that this will reduce the stretch by a logarithmic factor.

Acknowledgement. I would like to thank my advisor, David Peleg, for very helpful ideas, comments and observations. I also thank Michael Langberg and Liam Roditty for useful discussions.

References

1. Abraham, I., Chechik, S., Gavoille, C., Peleg, D.: Forbidden-set distance labels for graphs of bounded doubling dimension. In: PODC, pp. 192–200 (2010)
2. Alstrup, S., Brodal, G.S., Rauhe, T.: New Data Structures for Orthogonal Range Searching. In: Proc. 41st IEEE Symp. on Foundations of Computer Science (FOCS), pp. 198–207 (2001)
3. Awerbuch, B., Bar-Noy, A., Linial, N., Peleg, D.: Improved routing strategies with succinct tables. J. Algorithms, 307–341 (1990)
4. Awerbuch, B., Kutten, S., Peleg, D.: On buffer-economical store-and-forward deadlock prevention. In: Proc. INFOCOM, pp. 410–414 (1991)
5. Awerbuch, B., Peleg, D.: Sparse partitions. In: 31st FOCS, pp. 503–513 (1990)
6. Chechik, S., Langberg, M., Peleg, D., Roditty, L.: f-sensitivity distance oracles and routing schemes. In: 18th ESA, pp. 84–96 (2010)
7. Cohen, E.: Fast algorithms for constructing t-spanners and paths with stretch t. In: Proc. IEEE Symp. on Foundations of Computer Science, pp. 648–658 (1993)
8. Courcelle, B., Twigg, A.: Compact forbidden-set routing. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 37–48. Springer, Heidelberg (2007)
9. Cowen, L.J.: Compact routing with minimum stretch. J. Alg. 38, 170–183 (2001)
10. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications. Springer, Heidelberg (2008)
11. Duan, R., Pettie, S.: Connectivity oracles for failure prone graphs. In: Proc. ACM STOC (2010)
12. Eilam, T., Gavoille, C., Peleg, D.: Compact routing schemes with low stretch factor. J. Algorithms 46, 97–114 (2003)
13. Fraigniaud, P., Gavoille, C.: Memory requirement for universal routing schemes. In: 14th PODC, pp. 223–230 (1995)
14. Gavoille, C., Gengler, M.: Space-efficiency for routing schemes of stretch factor three. J. Parallel Distrib. Comput. 61, 679–687 (2001)
15. Gavoille, C., Peleg, D.: Compact and localized distributed data structures. Distributed Computing 16, 111–120 (2003)
16. Peleg, D.: Distributed computing: a locality-sensitive approach. SIAM, Philadelphia (2000)
17. Peleg, D., Upfal, E.: A trade-off between space and efficiency for routing tables. J. ACM 36(3), 510–530 (1989)
18. Thorup, M., Zwick, U.: Compact routing schemes. In: Proc. 13th ACM Symp. on Parallel Algorithms and Architectures (SPAA), pp. 1–10 (2001)
19. Thorup, M., Zwick, U.: Approximate distance oracles. J. ACM 52, 1–24 (2005)
20. Twigg, D.A.: Forbidden-set Routing. PhD thesis, University of Cambridge, King's College (2006)

Local Matching Dynamics in Social Networks*

Martin Hoefer

Dept. of Computer Science, RWTH Aachen University, Germany
mhoefer@cs.rwth-aachen.de

Abstract. We study stable marriage and roommates problems in graphs with locality constraints. Each player is a node in a social network and has an incentive to match with other players. The value of a match is specified by an edge weight. Players explore possible matches only based on their current neighborhood. We study convergence of natural better-response dynamics that converge to *locally stable matchings* – matchings that allow no incentive to deviate with respect to their imposed information structure in the social network. For every starting state we construct in polynomial time a sequence of polynomially many better-response moves to a locally stable matching. However, for a large class of oblivious dynamics including random and concurrent better-response the convergence time turns out to be exponential. In contrast, convergence time becomes polynomial if we allow the players to have a small amount of random memory, even for many-to-many matchings and more general notions of neighborhood.

1 Introduction

Matching problems are at the basis of many important assignment and allocation tasks encountered in economics and computer science. A prominent model for these scenarios are *stable matching* problems [13], as they capture the aspect of rationality and distributed control that is inherent in many assignment problems today. A variety of allocation problems in markets can successfully be analyzed within the context of two-sided stable matching, e.g., the assignment of jobs to workers [5, 14], organs to patients [19], or general buyers to sellers. In addition, stable marriage problems are an interesting approach to model a variety of distributed resource allocation problems in networks [3, 12, 18].

In this paper, we examine a dynamic variant of stable matching for collaboration in (social) networks without central coordination. Players are rational agents that are looking for partners for a joint activity or relationship, such as, e.g., to do sports, write a research paper, share an office, exchange data etc. Such problems are of fundamental interest in economics and sociology, and they serve as basic coordination tasks in computer networks with distributed control. We can capture these problems within the stable roommates problem, an extension of stable marriage that allows arbitrary pairs of players to be matched. A crucial aspect of ordinary stable marriage and roommates problems is that every player

* Supported by DFG grant Ho 3831/3-1.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 113–124, 2011.

knows the complete player set and can match arbitrarily. In contrast, for many of the examples above, we would not expect a player to be able to, e.g., write a research paper with any other player instantaneously. Instead, there are often restrictions in terms of knowledge and information that allow certain players to match up easily, while others need to get to know each other first before they can engage in a joint project. We incorporate this aspect by assuming that players are nodes in a static social network of existing *social links*. Each player strives to build a *matching edge* to another player. The network defines a dynamic information structure over the players, where we use a standard idea from social network theory called *triadic closure*: If a knows b and b knows c, then a and c are likely to meet and thus can engage in a joint project. When matching a to the 2-hop neighbor c, both players engage in a joint project and thus become more familiar with each other. In this case, triadic closure suggests that a learns about all direct neighbors of c, which can allow him to find a better matching partner among those players. More formally, at any point in time, we assume that each player can match only to *accessible* players, that is, players in the 2-hop neighborhood in the graph composed of social links and currently existing matching edges.

Traditionally, in the stable marriage problem we have sets of men and women, and each man (woman) has a full preference list over all women (men). Each man (woman) can be matched to exactly one woman (man), where all players would rather be matched than unmatched. A *blocking pair* is a pair of a man and a woman such that both prefer this match to their currently assigned partners (if any), and a *stable matching* is a matching that allows no blocking pair. It is well-known that in this case a stable matching always exists and can be found in polynomial time using the classic Gale-Shapley algorithm [11]. This does not hold for the extension to the stable roommates problem, where simple examples without stable matching exist. In this paper, we will focus on the prominent class of *correlated* [1, 2] (also called acyclic [18]) preferences. For the correlated stable roommates problem existence of a stable matching is guaranteed [1].

Contribution For many of the assignment and matching tasks in (social) networks that motivate our study, there is an inherent lack of information and coordination. Hence, we are interested in distributed dynamics that allow players with locality restrictions to reach a stable matching quickly. In particular, we consider convergence to *locally stable matchings* – matchings that allow no blocking pair of accessible players. We will focus on variants of sequential *best-response* and *better-response dynamics*, where in each round a blocking pair of accessible players (or *local blocking pair*) is allowed to deviate to establish their joint edge. It follows directly from results in, e.g., [1, 2] that our games are potential games, and thus all such dynamics are guaranteed to converge to a locally stable matching. This holds even for more general variants, in which each player can build up to $k > 1$ matching edges, or each player has access to all players within $\ell > 2$ hops in the graph.

After a formal definition of our model in Section 2, we show in Section 3 that in our basic game with $k = 1$ matching edges and lookahead $\ell = 2$ per player we can

always achieve fast convergence – for every game and every starting state there is a sequence of polynomially many better-response moves to a locally stable matching. While this shows that locally stable matchings are achievable, the sequence heavily relies on global knowledge of the graph. In contrast, oblivious dynamics without such knowledge like random or concurrent better-response need an exponential number of steps. When we turn to more general games with $k > 1$ or $\ell > 2$, there are even games and starting states such that *every* sequence of better-response moves to a locally stable matching is exponentially long. For the special case of stable marriage, we show that for general preference relations there are states, from which convergence might never be achieved. We also show improved results for special cases of the problem, especially for one-sided social networks and the job-market model of [5].

Perhaps surprisingly, instead of the usual structural aspects in social networks such as, e.g., low diameter or power-law degree distribution, a natural aspect resulting in polynomial time convergence is memory. In Section 5 we consider the case that every polynomial number of rounds each player remembers (uniformly at random) one of the players he had been matched to before. This seemingly small but powerful adjustment allows to show polynomial-time convergence for a variety of dynamics, for arbitrary $k \geq 1$ and $\ell \geq 2$. Finally, we also briefly touch upon the case when memory is considered as a cache in which a bounded number of good or recent matches are stored. Here we can again show an exponential lower bound for standard eviction strategies such as FIFO or LRU.

Related Work. There has been an enormous research interest in stable marriage and roommates problems over the last decades, especially in many-to-one matchings and preference lists with ties [6, 9, 14, 21]. For a general introduction to the topic, see standard textbooks [13, 21].

Recent theoretical work on convergence issues in ordinary stable marriage has focused on better-response dynamics, in which players sequentially deviate to blocking pairs. It is known that for stable marriage these dynamics can cycle [17]. On the other hand, there is always a sequence of polynomially many moves to a stable matching [20]. However, if the ordering of moves is chosen uniformly at random at each step, convergence time is exponential [2]. A prominent case with numerous applications [1, 4, 12, 18], in which fast convergence is achieved even by random dynamics, is correlated or weighted stable matching where each matched pair generates a benefit (or edge weight) for the incident players and the preferences of the players are ordered with respect to edge benefit [1, 18].

Local aspects of stable matching are of interest in distributed computing, e.g., communication complexity of distributed algorithms [16]. A localized version of stable marriage is analyzed in [10], where men and women are nodes in a graph and can only match to adjacent women or men, resp. Each player can only exchange messages with their neighbors and the goal is to design a local algorithm that computes an "almost" stable matching. Similar approaches to almost stable matchings in decentralized settings include, e.g., [7]. In addition, there exist online [15] and parallel algorithms [8] for stable marriage.

Our model of locality is similar to Arcaute and Vassilvitskii [5], who consider locally stable matchings in a specialized case of stable marriage. In their job-market game, there are firms that strive to hire workers. Social links exist only among workers, and each firm can match to k workers, but each worker only to one firm. They show that best-response dynamics converge almost surely and show several characterization results for locally stable matchings and the number of isolated firms after a run of a local variant of the Gale-Shapley algorithm. In this paper we greatly extend their results on convergence of dynamics.

2 Model and Initial Results

In our model we are given a social *network* $N = (V, L)$, where V is a set of n *players* and $L \subseteq V \times V$ is a set of undirected and fixed social *links*. In addition, we have a set E of undirected *potential matching edges*, where we denote $m = |E|$. Each edge $e \in E$ has a *benefit* $b(e) > 0$. A *state* $M \subseteq E$ of the game contains for each player at most k incident matching edges, i.e., each player can be involved in up to $k \geq 1$ matching edges simultaneously. Unless specified otherwise we will assume throughout that $k = 1$. The *utility* or *welfare* of a player u in state M is $\sum_{\{u,v\} \in M} b(\{u, v\})$ if he is matched to at least one player and 0 otherwise. The restriction of E to a subset of all edges will be mostly for convenience and clarity of presentation. Our lower bounds can be adjusted to allow every pair as matching edge using minor technical adjustments that blow up the network using dummy players and for $e \notin E$ use benefits that are either extremely tiny (to keep an edge from becoming a blocking pair) or extremely large (to "hard-wire" an edge and thereby change the matching incentives). Details are left for the full version of this paper.

In a state M two players u and v are *accessible* if there is a path of length at most ℓ in the *access graph* $G = (V, L \cup M)$. We call ℓ the *lookahead* of the game and, unless stated otherwise, focus on the case of triadic closure and $\ell = 2$. An edge $e = \{u, v\} \in E$ is called a *local blocking pair* for M if u and v are accessible, and if for each of u and v either (a) the player has less than k matching edges in M or (b) at least one incident edge $e' \in M$ has $b(e') < b(e)$. Hence, for a local blocking pair the accessible players both strictly increase their welfare by either adding e or replacing e' by e. In the latter case, the replaced edges are removed from M. We call $b(e)$ the benefit of the local blocking pair. A *locally stable matching* is a state M for which there is no local blocking pair.

We consider round-based improvement dynamics that lead to locally stable matchings. In a *local improvement move* we pick a local blocking pair and allow the involved players to deviate to their joint edge, thereby potentially removing other edges. We consider sequential processes that in every round implement a local improvement move. For *(random) best-response dynamics*, we pick in each round deterministically (uniformly at random) a local blocking pair of maximum benefit. As *(random) better-response dynamics* we term all sequential dynamics that in each round pick one local blocking pair in a deterministic (uniformly at random) way. Finally, for *concurrent better- (best-)response dynamics* we assume that every player picks uniformly at random from the local blocking pairs

he is involved in (and that are of maximum benefit among those available for him). For every local blocking pair that is chosen by both incident players the corresponding edge is built concurrently.

In addition, we also consider better-response dynamics with memory. For a dynamics with *random memory* we assume that each player at some point recalls a player that he had been matched to before. In particular, let M_v be the set of players that $v \in V$ has been matched with at some point during the history of play. We assume that, in expectation, every T rounds a player v remembers a player u chosen uniformly at random from M_v, and u and v become temporarily accessible in this round. For dynamics with *cache memory* we assume that each player has a cache in which he can keep up to r players previously matched to him. A pair of players then is accessible if and only if they are currently at hop distance at most ℓ in G or one player is present in the cache of the other player.

Our games reduce to the ordinary correlated stable roommates problem for $L = V \times V$. Thus, every ordinary stable matching is a locally stable matching for every network N. Note that an ordinary stable matching can be computed in time $O(n \log n)$ by repeated addition of an edge for a blocking pair with maximum benefit. Hence, centralized computation of a locally stable matching is trivially in P, for every $k \geq 1$ and every $\ell \geq 2$. For ordinary correlated stable roommates even random best-response dynamics converge in polynomial time [2].

3 Convergence of Better-Response Dynamics

In this section we consider the duration of sequential and concurrent improvement dynamics. Even when we restrict to accessible players, a new edge built due to a local blocking pair destroys only edges of strictly smaller benefit. This implies the existence of a lexicographical potential function. Hence, both sequential and concurrent better-response dynamics are always guaranteed to converge to a locally stable matching. Moreover, for our standard case with $k = 1$ and $\ell = 2$ the first result shows that we can always achieve fast convergence.

Theorem 1. *For any state there is a sequence of $O(n \cdot m^2)$ local improvement moves that lead to a locally stable matching. The sequence can be computed in polynomial time.*

Proof. Consider a game with a given network N and a state specified by a set M of existing matching edges. A local blocking pair with edge e falls in one of two categories (1) the players are at distance at most 2 in N or (2) the players are connected via one existing matching edge e' and one link from L. If e falls in category (2), edge e' gets destroyed, and we can think of the edge moving from e' to e. This is the motivation for our main tool in this proof, the *edge movement graph* G_{mov}. This vertex set of this graph is E. Each vertex $\{u, v\} \in E$ has a corresponding vertex weight $b(\{u, v\})$. If u and v are at distance at most 2 in N, their vertex in G_{mov} is called *starting point*. We denote the set of all starting points by S.

There are two kinds of edges in G_{mov}, *movement edges* and *domination edges*. For every triple of players $u, v, w \in V$ we introduce a directed movement edge in

G_{mov} from $\{u, v\}$ to $\{u, w\}$ when $\{u, v\}, \{u, w\} \in E$, $\{v, w\} \in L$ and $b(\{u, v\}) < b(\{u, w\})$. When edge $\{u, v\}$ exists, then u and w get accessible. If there is no other matching edge in the system, $\{u, w\}$ becomes a local blocking pair which is expressed by the movement edge. Note that movement edges induce a DAG. Domination edges describe the fact that potentially the existence of one matching edge prohibits creation of another one. For all pairs $\{u, v\}$ and $\{u, w\}$ in G_{mov} we introduce a directed domination edge from $\{u, v\}$ to $\{u, w\}$ when $b(\{u, v\}) \geq b(\{u, w\})$. In this case $\{u, w\}$ is *dominated* by $\{u, v\}$. If the $b(\{u, v\}) > b(\{u, w\})$, $\{u, w\}$ is *strictly dominated* by $\{u, v\}$.

Consider the subgraph of G_{mov} that is reachable from $S \cup M$ via movement edges. If a pair is not in this subgraph, there is no sequence of local improvement moves starting from M that establishes an edge between these players. We prune the graph and consider only the subgraph reachable from $S \cup M$.

A state M of the game can be seen as a marking of the vertices in G_{mov} that correspond to edges in M. A local improvement move from M can only happen if some marked vertex p has an outgoing movement edge to another vertex p' (as $k = 1$, p' must be unmarked). This represents a feasible local improvement move only if p' is currently *undominated*, i.e., has no incoming domination edge from a marked vertex. We will describe how to migrate the markings along movement edges to reach a locally stable matching in a polynomial number of rounds.

Phase 1: In phase 1 we move existing markings without introducing new ones. As long as it is possible, we arbitrarily move an existing marking along a movement edge to an undominated vertex one at a time. Note that the set of dominated vertices changes in every round. In particular, a marking is deleted if it becomes dominated in the next round (because a dominating neighbor with higher benefit becomes marked). Thus, in this process the number of markings only decreases. In addition, for each marking, the vertex weight of the location only increases. Due to acyclicity, a particular marking can visit each node of G_{mov} only once, thus the phase ends after at most $O(n \cdot m)$ many rounds.

Phase 2: In phase 2 we try to improve existing markings even further by introducing new ones and moving them via undominated paths to strictly dominating vertices. In particular, for a marked vertex $\{u, v\}$ in G_{mov} we do the following. We drop all currently dominated vertices from consideration. Now we try to find a path of movement edges from a starting point to a vertex that strictly dominates $\{u, v\}$. If there is such a path, we can introduce a new matching edge via a new marking at the starting point and move it along the path to the dominating vertex. Due to the fact that none of the path vertices are dominated, all the moves are local improvement moves in the game. All markings that become dominated during this process are removed. This also includes in the last step our original marking at $\{u, v\}$, which we can think of as moving to the dominating vertex. After this, we try to improve all markings by a restart of phase 1. We keep executing this procedure until this kind of improvement is impossible for every remaining marking.

Phase 2 can be seen as an extension of phase 1. Overall, we keep decreasing the number of markings, and each surviving marking is increased in terms of

vertex weight. However, each such increase might now require $O(m)$ steps, which increases the number of rounds to at most $O(n \cdot m^2)$.

Phase 3: After phase 2, none of the remaining markings can be (re)moved, neither by moving the marking along a movement edge to another undominated vertex, nor by a sequence of moves that lead to creation of a marking at a dominating vertex (verify that otherwise phase 2 would not have ended). Hence, these edges are stable, and we call the incident players *stabilized*. They will not become part of any local blocking pair in the remaining process. We drop all these players from the game and adjust G_{mov} by dropping all vertices including at least one stabilized player. Finally, we now iteratively construct the matching edge of largest possible benefit until no more edge can be constructed. In particular, we consider the reduced G_{mov} (which is now completely unmarked) and find a vertex with largest benefit that is reachable from a starting point. We then establish the corresponding edge by moving a new marking along the path. There is no path to any edge with strictly larger benefit, and no player will get an incentive to remove this edge. In particular, after removing edges and incident players, no such path will become possible at a later point in the process. Hence, iterative application of this final step completes the locally stable matching. This phase terminates after $O(n \cdot m)$ rounds in total.

The construction and adjustment of G_{mov} can be trivially be done in polynomial time. For finding the sequences of local improvement moves we essentially require only algorithms for DFS or BFS in DAGs. □

The computation of the short sequence in the previous theorem relies heavily on identification of proper undominated paths in the edge movement graph. If we instead consider random or concurrent dynamics that do not rely on such global information, convergence time can become exponential.

Theorem 2. *For every $b \in \mathbb{N}$ there is a game with $n \in \Theta(b)$, in which (random) best-response and random and concurrent better-response dynamics starting from the state $M = \emptyset$ need $\Omega(1.5^b)$ steps in expectation to converge to a locally stable matching.*

Proof. We construct our game based on a structure we call an "edge trap". We concatenate b edge traps as shown in Fig. 1. In a trap i there is a starting edge e_i. We assume that initially there is no matching edge and lower bound the number of times that e_i must be created to lead to construction of e_{i+1}. Upon creation of e_i, we assume there are exactly two local blocking pairs that both destroy e_i – either create e^u or e^l, which implies $b(e_i) < \min\{b(e^u), b(e^l)\}$. If e^u is formed, $\{w_i, v_{i+1}\}$ becomes a local blocking pair with e^c destroying edge e^u (i.e., $b(e^u) < b(e^c)$). At this point, both w_i and v_{i+1} are happy with their matching choice. Now suppose e_i is created again, then player w_i will not form e^u. In this case, v_i matches to u_{i+1} via e^l. Now v_{i+1} has an incentive to drop e^c and match with u_{i+1}, because $b(e^c) < b(e_{i+1})$.

By sequential concatenation of edge traps we can make the dynamics simulate a counter with b bits. In particular, we assume there are b traps attached sequentially, where in the first trap players u_1 and v_1 are connected with a path

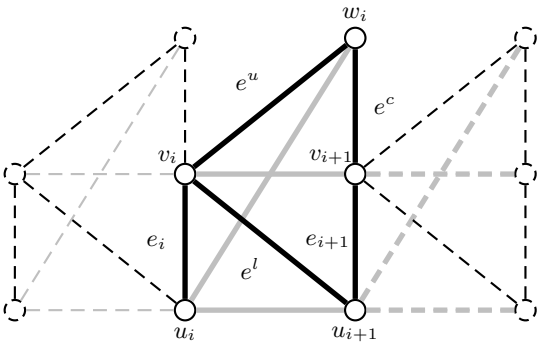


Fig. 1. Structure of edge traps in the lower bound of Theorem 2. Social links are indicated in gray, possible matching edges are drawn in black.

of distance 2 using a dummy player. For the last pair of nodes u_{b+1} and v_{b+1} we assume there is also the final edge e_{b+1}. Bit i is set if and only if edge e^c in trap i is created. We start the dynamics with the empty matching $M = \emptyset$, so the counter is 0. Note that N is a tree with a long path and satellite nodes.

First consider best-response dynamics, for which we set $b(e^u) > b(e^l)$ in every trap. Then creation of e_1 implies creation of e^c in the first trap, i.e., increase of the counter by 1. At this point, the edge is trapped, and the only local improving move is to create e_1 again. This leads to destruction of e^c in the first trap, creation of e^c in the second trap, and thus an increase in the bit counter of 1. Continuing in this fashion we see that every creation of e_1 leads to a state that represents an increase of the bit counter by 1. Thus, to create e_{b+1}, the edge of largest benefit, the dynamics needs $\Omega(2^b)$ many creations of e_1. Note that in each step there is a unique local blocking pair of maximum benefit. This proves the lower bound for both deterministic and random best-response dynamics.

Note that for each state reached during the dynamics, only one local blocking pair except (u_1, v_1) can apply their deviation. In particular, e^u and e^l cannot be created simultaneously. Hence, as long as the creation of e^u is sufficiently likely in every trap, we can trap enough edges that are subsequently destroyed and show a similar lower bound. This is the case, in particular, for random and concurrent better-response dynamics. Note that every locally stable matching must contain the edge e_{b+1}. We now bound the number of times we have to create e_1 until e_{b+1} forms. Consider the first trap, and suppose edge e_1 is created. We now follow this edge moving through the trap. With probability at most 1/2, the edge moves directly to e^l and arrives at e_2. With probability at least 1/2, the edge gets stuck in e^c, which implies that the edge is destroyed and the next edge created at e_1 arrives at e_2 with probability 1. Thus, to create a single edge at e_2 we have to create e_1 an expected number of 1.5 times. The same is true for e_i and e_{i+1} in every trap i. Thus, due to the sequential nature of the gadget, we need an expected number of $\Omega(1.5^b)$ creations of e_1 to create edge e_{b+1}. □

The previous theorem applies similarly to a wide variety of better-response dynamics. The main property is that whenever e_i exists and both e^u and e^l are available for creation, then the creation of e^u is always at least as likely as that of e^l. Observe that the construction allows to set $b(e^u) < b(e^l)$ or vice versa. This allows to satisfy the property for many dynamics that make "oblivious" choices by picking local blocking pairs based only on their benefits but not their structural position or the history of the dynamics. An even stronger lower bound applies when we increase lookahead or matching edges per player.

Theorem 3. *Let $k \geq 2$ or $\ell \geq 3$, then for every $b \in \mathbb{N}$ there is a game with $n \in \Theta(b \cdot k \cdot \ell)$ and a starting state M such that every sequence of local improvement moves from M to a locally stable matching has length $\Omega(2^b)$.*

By embedding the lower bound structures from the previous proofs into larger graph structures of dummy vertices, we can impose a variety of additional properties on the network N that are often considered in the social network literature. For example, to obtain a small diameter simply add a separate source and connect each vertex from the gadgets via ℓ dummy vertices to the source. As these new vertices have no matching edges, the lower bounds hold accordingly for graphs of diameter $\Theta(\ell)$. Note that for a diameter of exactly ℓ we obtain the ordinary correlated stable roommates problem, for which polynomial time convergence is guaranteed. As mentioned earlier there also exist simple adjustments using dummy vertices and tiny and extremely large benefits to adjust the results to $E = V \times V$ with all possible matching edges.

Corollary 1. *Theorems 2 and 3 continue to hold even if $diam(N) \in \Theta(\ell)$.*

4 Two-Sided Matching and Stable Marriage

In this section we consider the bipartite case of stable marriage. In accordance with [5] we use the interpretation of a set W of "workers" matching to a set F of "firms". We use $n_w = |W|$ and $n_f = |F|$. In general, the social network $N = (F \cup W, L)$ can be arbitrary, and the set of possible matching edges is $E = W \times F$. Each worker (firm) has an arbitrary preference relation over firms (workers). In contrast to ordinary stable marriage, in our localized variant convergence of any better-response dynamics can be impossible.

Theorem 4. *For stable marriage with general preferences, there are games and starting states M such that no locally stable matching can be reached by any sequence of local improvement moves from M, even if N is connected.*

For weighted matching with correlated preferences, Theorem 1 applies and shows existence of a short sequence. A central assumption of [5] is that every worker $w \in W$ has the same preference list over F, and every firm $f \in F$ has the same preference list over W. We will refer to this case as *totally uniform preferences*. As a generalization we consider *worker-uniform* preferences, where we assume that only the preferences of all workers are the same, while firms have arbitrary

preferences. *Firm-uniform* preferences are defined accordingly. For totally uniform preferences we can number firms and workers increasingly from least to most preferred in their respective global preference list. For edge $e_{ij} = (w_i, f_j)$ we define a benefit $b(e_{ij}) = j \cdot n_w + i$. Intuitively, here best-response dynamics give preference to local blocking pairs of the most preferred firm, which can be changed to worker by using $b(e_{ij}) = i \cdot n_f + j$ throughout. For worker-uniform preferences we let the numbering of workers be arbitrary. For $e_{ij} = (w_i, f_j)$ we define benefit $b(e_{ij}) = j \cdot n_w + i_j$, when worker w_i is ranked at the i_j-th last position in the preference list of firm f_j. For firm-uniform preferences the same idea can be used by exchanging the roles of firms and workers. This shows that all these cases are classes of correlated stable matching problems. Our first result is that even for totally uniform preferences, convergence of best-response dynamics can be slow.

Theorem 5. *For every $b \in \mathbb{N}$, there is a game with totally uniform preferences, $n_f, n_w \in \Theta(b)$ and a starting state M such that (random) best-response dynamics from M to a locally stable matching take (expected) time $\Omega(2^b)$.*

A case in which such preferences can still lead to quick convergence is in the *job-market game* considered in [5]. Note that for the following theorem we only need worker-uniform preferences.

Theorem 6. *For every job-market game with worker-uniform preferences, in which each firm can create up to $k \geq 1$ matching edges, (random) best-response dynamics converge to a locally stable matching from every starting state in (expected) time $O(n_f \cdot n_w \cdot k)$.*

The result directly extends to random and concurrent better-response dynamics when spend an additional factor of $n_f \cdot n_w$. In contrast, if we consider firm-uniform preferences, a lower bound for best-response dynamics can be shown.

Theorem 7. *For every $b \in \mathbb{N}$ and $k \geq 2$, there is a job-market game with firm-uniform preferences, $n_f \in \Theta(b)$, $n_w \in \Theta(b \cdot k)$ and a starting state M such that (random) best-response dynamics from M to a locally stable matching take (expected) time $\Omega(2^b)$.*

5 Dynamics with Memory

In this section we consider sequential and concurrent better-response dynamics with memory. Our first result is a polynomial time bound for random memory that holds accordingly for random and concurrent better-response dynamics.

Theorem 8. *For every $k, \ell \in \mathbb{N}$ and every game, in which each player can create up to k matching edges and has lookahead ℓ, (random) best-response dynamics with random memory converge to a locally stable matching from every starting state in (expected) time $O(n^2 \cdot m^2 \cdot k \cdot T)$.*

Proof. The main insight is that all the above mentioned dynamics can rely on the information in the random memory to steer the convergence towards a locally stable matching. Let us consider the dynamics in phases. Phase t begins after the dynamics has created t mutually different matching edges at least once (including the ones in the starting state). Let E_t be the set of edges which have been created at least once when entering phase t. During phase t no new edge is created for the first time. The main insight is that in a phase all the dynamics converge in polynomial time. In particular, consider an edge $e \in E_t$ that represents a (global) blocking pair and has maximum benefit. A round in which such an edge is available for creation appears in expectation at most every $n \cdot T$ rounds. When it is available for creation all the dynamics mentioned above will create it with probability at least $\Omega(1/m)$. The deterministic best-response dynamics might create a different edge of maximum benefit, but it has to pick e after at most m such rounds unless e stops being a blocking pair. Thus, after an expected number of $O(n \cdot m \cdot T)$ rounds, e is either created or stops being a blocking pair. In the former case, it will not get removed in phase t again, in the latter case other incident edges from E_t of maximum benefit were created that are not removed in phase t again. Hence, at least one of the incident players has an edge that he is not willing to remove in phase t. By repeated application of this argument, we see that after at most $n \cdot k$ of these steps all blocking pairs have been removed. Thus, after an expected number of $O(n^2 \cdot m \cdot k \cdot T)$ rounds the phase ends in a matching that is stable with respect to E_t. Finally, we note that there are at most m many phases to be considered. □

With random memory no previous matching edge can be completely forgotten during the dynamics. Can we allow players to forget some previous matches and still obtain fast convergence with local dynamics? Towards this end, we here consider two natural examples and show that delayed forgetting of previous matches can be harmful to convergence. In particular, we consider best-response dynamics with cache memory and largest-benefit, FIFO or LRU eviction strategies.

Corollary 2. *If each player keeps only the r best matches in his cache, then for every $b \in \mathbb{N}$ there is a game with $n \in \Theta(b \cdot r)$, in which best-response dynamics starting from the state $M = \emptyset$ need $\Omega(2^b)$ steps to converge to a locally stable matching.*

Theorem 9. *If each player keeps only the r most recent matches in his cache, then for every $b \in \mathbb{N}$ there is a game with $n \in \Theta(b^2 \cdot r)$, in which best-response dynamics starting from the state $M = \emptyset$ need $\Omega(2^b)$ steps to converge to a locally stable matching.*

Acknowledgments

The author would like to thank Siddharth Suri and Heiko Röglin for numerous insightful discussions on the topic.

References

1. Abraham, D., Levavi, A., Manlove, D., O'Malley, G.: The stable roommates problem with globally ranked pairs. Internet Math. 5(4), 493–515 (2008)
2. Ackermann, H., Goldberg, P., Mirrokni, V., Röglin, H., Vöcking, B.: Uncoordinated two-sided matching markets. SIAM J. Comput. 40(1), 92–106 (2011)
3. Akkaya, K., Guneydas, I., Bicak, A.: Autonomous actor positioning in wireless sensor and actor networks using stable-matching. Intl. J. Parallel, Emergent and Distrib. Syst. 25(6), 439–464 (2010)
4. Anshelevich, E., Hoefer, M.: Contribution games in social networks. In: Proc. 18th European Symposium on Algorithms (ESA), vol. 1, pp. 158–169 (2010)
5. Arcaute, E., Vassilvitskii, S.: Social networks and stable matchings in the job market. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 220–231. Springer, Heidelberg (2009)
6. Echenique, F., Oviedo, J.: A theory of stability in many-to-many matching markets. Theoretical Economics 1(2), 233–273 (2006)
7. Eriksson, K., Häggström, O.: Instability of matchings in decentralized markets with various preference structures. Intl. J. Game Theory 36(3-4), 409–420 (2008)
8. Feder, T., Megiddo, N., Plotkin, S.: A sublinear parallel algorithm for stable matching. Theoret. Comput. Sci. 233(1-2), 297–308 (2000)
9. Fleiner, T.: A fixed-point approach to stable matchings and some applications. Math. Oper. Res. 28(1), 103–126 (2003)
10. Floréen, P., Kaski, P., Polishchuk, V., Suomela, J.: Almost stable matchings by truncating the gale-shapley algorithm. Algorithmica 58(1), 102–118 (2010)
11. Gale, D., Shapley, L.: College admissions and the stability of marriage. Amer. Math. Monthly 69(1), 9–15 (1962)
12. Goemans, M., Li, L., Mirrokni, V., Thottan, M.: Market sharing games applied to content distribution in ad-hoc networks. IEEE J. Sel. Area Comm. 24(5), 1020–1033 (2006)
13. Gusfield, D., Irving, R.: The Stable Marriage Problem: Structure and Algorithms. MIT Press, Cambridge (1989)
14. Kelso, A., Crawford, V.: Job matchings, coalition formation and gross substitute. Econometrica 50, 1483–1504 (1982)
15. Khuller, S., Mitchell, S., Vazirani, V.: On-line algorithms for weighted bipartite matching and stable marriages. Theoret. Comput. Sci. 127(2), 255–267 (1994)
16. Kipnis, A., Patt-Shamir, B.: On the complexity of distributed stable matching with small messages. Distributed Computing 23(3), 151–161 (2010)
17. Knuth, D.: Marriages Stables et leurs relations avec d'autres problemes combinatoires. Les Presses de l'Université de Montréal (1976)
18. Mathieu, F.: Self-stabilization in preference-based systems. Peer-to-Peer Netw. Appl. 1(2), 104–121 (2008)
19. Roth, A., Sönmezc, T., Ünverd, M.U.: Pairwise kidney exchange. J. Econ. Theory 125(2), 151–188 (2005)
20. Roth, A., Sotomayor, M.O.: Two-sided Matching: A study in game-theoretic modeling and analysis. Cambridge University Press, Cambridge (1990)
21. Roth, A., Vate, J.H.V.: Random paths to stability in two-sided matching. Econometrica 58(6), 1475–1480 (1990)

Regular Languages of Words over Countable Linear Orderings*

Olivier Carton[1], Thomas Colcombet[2], and Gabriele Puppis[3]

[1] University Paris Diderot, LIAFA
{olivier.carton,thomas.colcombet}@liafa.jussieu.fr
[2] CNRS/LIAFA
[3] Department of Computer Science, Oxford University
gabriele.puppis@cs.ox.ac.uk

Abstract. We develop an algebraic model for recognizing languages of words indexed by countable linear orderings. This notion of recognizability is effectively equivalent to definability in monadic second-order (MSO) logic. The proofs also imply the first known collapse result for MSO logic over countable linear orderings.

1 Introduction

This paper continues a long line of research aiming at understanding the notions of regularity for languages of infinite objects, i.e., of infinite words and trees. This research results both in decision procedures for the *monadic second-order* (*MSO*) *logic* and in a fine comprehension of the mechanisms involved in different models of recognition. More specifically, the paper answers the following interrogations:

> *What is a good notion of regularity for languages of words indexed by countable linear orderings? Is it equivalent to definability in MSO? What are the correct tools for studying this notion?*

Several results, in particular in terms of decidability, partially answered the above questions (see related work below). Our study gives a deeper insight in the phenomena, e.g. answering (positively) the following previously open question:

> *Does there exist a collapse result for MSO logic over countable linear orders, as Büchi's result shows a collapse of MSO logic to its existential fragment for words indexed by ω?*

The central objects in this paper are *words* indexed by countable linear orderings, i.e., total orders over countable sets together with functions mapping elements to letters in some finite alphabet. Languages are just sets of countable words and MSO logic gives a formalism for describing such languages in terms of formulas involving quantification over elements and sets of elements (a formula naturally defines the language of all words that makes the formula true).

* Supported by the Anr 2007 JCJC 0051 Jade and Anr 2010 Blan 0202 02 Frec.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 125–136, 2011.

Related work. Büchi initiated the study of MSO using the tools of language theory. He established that every language of ω-words (i.e., the particular case of words indexed by the ordinal ω) definable in MSO is effectively recognized by a suitable form of automaton [4]. A major advance has been obtained by Rabin, who extended this result to infinite trees [8]. One consequence of Rabin's result is that MSO is decidable over the class of all countable linear orderings. Indeed, every linear ordering can be seen as a set of nodes of the infinite tree, with the order corresponding to the infix ordering on nodes. Another proof of the decidability of the MSO theory of countable orders has been given by Shelah using the composition method [12]. This is an automaton-free approach to logic based on syntactic operations on formulas and inspired from Feferman and Vaught [6]. The same paper of Shelah is also important for another result it contains: the undecidability of the MSO theory of the real line (the reals with order). However, for ω-words as for infinite trees, the theory is much richer than simply the decidability of MSO. In particular, MSO is known to be equivalent to several formalisms, such as automata and, in the ω-word case, regular expressions and some forms of algebras, which give a very deep insight in the structure of languages. The decidability proof for MSO does not provide such an understanding.

A branch of research has been pursued to raise the equivalence between logic, automata, and algebra to infinite words beyond ω-words. In [3], Büchi introduced ω_1-automata on transfinite words to prove the decidability of MSO logic for ordinals less than ω_1. Besides the usual transitions, ω_1-automata are equipped with limit transitions of the form $P \to q$, with P set of states, which are used in a Muller-like way to process words indexed over ordinals. Büchi proved that his automata have the same expressive power as MSO logic for ordinals less than ω_1. The key ingredient is the closure under complementation of ω_1-automata.

In [2], ω_1-automata have been extended to $\diamond$-automata by introducing limit transitions of the form $q \to P$ to process words over linear orderings. In [10], $\diamond$-automata are proven to be closed under complementation with respect to countable and scattered orderings. This last result implies that $\diamond$-automata have the same expressive power as MSO logic over countable and scattered orderings [1]. However, it was already noticed in [1] that $\diamond$-automata are strictly weaker than MSO logic over countable (possibly non-scattered) linear orderings: indeed, the closure under complementation fails as there is an automaton that accepts all words with non-scattered domains, whereas there is none for scattered words.

In this paper, we unify those branches of research. We provide an algebraic framework and a notion of recognizability which happens to be equivalent to the definability in MSO logic. Our approach both extends the decidability approach of Rabin and Shelah, and provides new results concerning the expressive power of MSO logic over countable linear orders. In preliminary versions of this work, we devised an equivalent automaton model. This notion is less natural and it is not presented in this short paper.

Structure of the paper. After the preliminaries in Section 2, we present $\circ$-algebras and the corresponding tools and results in Section 3. In Section 4 we translate MSO formulas to $\circ$-algebras and in Section 5 we establish the converse.

2 Preliminaries

Linear orderings. A *linear ordering* $\alpha = (X, <)$ is a non-empty set X equipped with a total order $<$. Two linear orderings have same *order type* if there is an order-preserving bijection between their domains. We denote by ω, ω^*, ζ, η the order types of $(\mathbb{N}, <)$, $(-\mathbb{N}, <)$, $(\mathbb{Z}, <)$, $(\mathbb{Q}, <)$, respectively. Unless strictly necessary, we do not distinguish between a linear ordering and its order type. A *sub-ordering* of α is a subset I of α equipped with the same ordering relation (we denote it by $\alpha|_I$). Given two subsets I, J of α, we write $I < J$ iff $x < y$ for all $x \in I$ and all $y \in J$. A subset I of α is said to be *convex* if for all $x, y \in I$ and all $z \in \alpha$, $x < z < y$ implies $z \in I$. A linear ordering α is *dense* if for all $x < y \in \alpha$, there is $z \in \alpha$ such that $x < z < y$. It is *scattered* if none of its sub-orderings is both dense and non-singleton.

The *sum* $\alpha_1 + \alpha_2$ of two linear orderings $\alpha_1 = (X_1, <_1)$ and $\alpha_2 = (X_2, <_2)$ (up to renaming, assume that X_1 and X_2 are disjoint) is the linear ordering $(X_1 \uplus X_2, <)$, where $<$ coincides with $<_1$ on X_1, with $<_2$ on X_2, and, furthermore, it satisfies $X_1 < X_2$. More generally, given a linear ordering $\alpha = (X, <)$ and, for each $i \in X$, a linear ordering $\beta_i = (Y_i, <_i)$ (assume that the sets Y_i are pairwise disjoint), we denote by $\sum_{i \in \alpha} \beta_i$ the linear ordering $(Y, <')$, where $Y = \bigcup_{i \in X} Y_i$ and, for every $i, j \in X$, every $x \in Y_i$, and every $y \in Y_j$, $x <' y$ iff either $i = j$ and $x <_i y$ hold or $i < j$ holds.

Additional material on linear orderings can be found in [11].

Condensations. A standard way to prove properties of linear orderings is to decompose them into basic components (e.g., finite sequences, ω-sequences, ω^*-sequences, and η-orderings). This can be done by exploiting the notion of condensation. Precisely, a *condensation* of a linear ordering α is an equivalence relation $\sim$ over it such that for all $x < y < z$ in α, $x \sim z$ implies $x \sim y \sim z$ (this is equivalent to enforcing the condition that every equivalence class of $\sim$ is a convex subset). The ordering relation of α induces a corresponding total order on the *quotient* $\alpha/\sim$, which is called the *condensed order*. Conversely, any partition C of α into convex subsets induces a condensation $\sim_C$ such that $x \sim y$ iff x and y belong to the same convex subset $I \in C$.

Words and languages. We use a generalized notion of word, which coincides with the notion of labeled linear ordering. Given a linear ordering α and a finite alphabet A, a *word over A with domain α* is a mapping of the form $w : \alpha \to A$. Hereafter, we shall always consider words up to isomorphism of the domain, unless specifically required. Moreover, we only consider words of countable domains. The set of all words (of countable domain) over an alphabet A is denoted by A°. Given a word w with domain α and a non-empty subset I of α, we denote by $w|_I$ the *sub-word* resulting from the restriction of the domain of w to I. Furthermore, if I is convex, then $w|_I$ is said to be a *factor* of w.

Certain words will play a crucial role in the sequel. In particular, a word $w : \alpha \to A$ is said to be a *perfect shuffle of A* if (i) the domain α is isomorphic to $\mathbb{Q}$ and (ii) for every symbol $a \in A$, the set $w^{-1}(a) = \{x \in \alpha \mid w(x) = a\}$

is dense in α. Recall that $\mathbb{Q}$ is the unique, up to isomorphism, countable non-singleton dense linear ordering with no end-points. Likewise, for every finite set A, there is a unique, up to isomorphism, perfect shuffle of A.

Given two words $u : \alpha \to A$ and $v : \beta \to A$, we denote by uv the word with domain $\alpha + \beta$ where each position $x \in \alpha$ (resp., $x \in \beta$) is labeled by $u(x)$ (resp., $v(x)$). The concatenation of words is easily generalized to the infinite concatenation $\prod_{i\in\alpha} w_i$, where α is a linear ordering and each w_i has domain β_i, the result being a word with domain $\sum_{i\in\alpha} \beta_i$. We also define the *shuffle* of a tuple of words $w_1, ..., w_k$ as the word $\{w_1, ..., w_k\}^\eta = \prod_{i\in\mathbb{Q}} w_{f(i)}$, where f is the unique perfect shuffle of $\{1, ..., k\}$ with domain $\mathbb{Q}$.

A *language* is a set of words over a certain alphabet. For some technical reasons, it is convenient to avoid the presence of the empty word ε in a language. Thus, unless otherwise specified, we use the term word to mean a labeled, countable, non-empty linear ordering. The operations of juxtaposition, ω-iteration, ω^*-iteration, shuffle, etc. are extended to languages in the obvious way.

3 Semigroups and Algebras for Countable Linear Orderings

This section is devoted to the presentation of algebraic objects suitable for defining a notion of recognizable $\circ$-languages. As it is already the case for ω-words, our definitions come in two flavors, $\circ$-semigroups (corresponding to ω-semigroups) and $\circ$-algebras (corresponding to Wilke-algebras). We prove the equivalence of the two notions when the underlying set is finite.

Countable products. The objective is to have a notion of products indexed by countable linear orderings, and possessing several desirable properties (in particular, generalized associativity and existence of finite presentations).

Definition 1. *A* (generalized) product *over a set S is a function π from S° to S such that, for every $a \in S$, $\pi(a) = a$ and, for every word u and every condensation $\sim$ of its domain,*

$$\pi(u) \;=\; \pi\Big(\prod\nolimits_{I\in\alpha/\sim} \pi(u|_I)\Big) \qquad \text{(generalized associativity)}$$

The pair $\langle S, \pi\rangle$ is called a $\circ$-semigroup.

As an example, the operation $\prod$ is a generalized product over A°. Hence, $\langle A^\circ, \prod\rangle$ is a $\circ$-semigroup (in particular, it is the *free* $\circ$-semigroup generated by A).

A *morphism* from a $\circ$-semigroup (S, π) to another $\circ$-semigroup (S', π') is a mapping $\varphi : S \to S'$ such that, for every word $w : \alpha \to S$, $\varphi(\pi(w)) = \pi'(\widetilde{\varphi}(w))$, where $\widetilde{\varphi}$ is the component-wise extension of φ to words. A $\circ$-language $L \subseteq A^\circ$ is called *recognizable by $\circ$-semigroups* if there exists a morphism φ from $\langle A^\circ, \prod\rangle$ to some finite semigroup $\langle S, \pi\rangle$ (here finite means that S is finite) such that $L = \varphi^{-1}(F)$ for some $F \subseteq S$ (equivalently, $\varphi^{-1}(\varphi(L)) = L$).

Recognizability by $\circ$-semigroup has the expressive power we aim at, however, the product π requires to be represented, a priori, by an infinite table. This is not

usable as it stands for decision procedures. That is why, given a finite $\circ$-semigroup $\langle S, \pi\rangle$, we define the following (finitely presentable) algebraic operators:

- $\cdot : S^2 \to S$, mapping a pair of elements $a, b \in S$ to the element $\pi(ab)$,
- $\tau : S \to S$, mapping an element $a \in S$ to the element $\pi(a^\omega)$,
- $\tau^* : S \to S$, mapping an element $a \in S$ to the element $\pi(a^{\omega^*})$,
- $\kappa : \mathscr{P}(S)\backslash\{\emptyset\} \to S$, mapping a non-empty set $\{a_1, ..., a_k\}$ to $\pi\big(\{a_1, ..., a_k\}^\eta\big)$.

One says that $\cdot, \tau, \tau^*$ and κ are *induced by* π. From now on, we shall use the operator $\cdot$ with infix notation (e.g., $a \cdot b$) and the operators τ, τ^*, and κ with superscript notation (e.g., a^τ, $\{a_1, ..., a_k\}^\kappa$). The resulting algebraic structure $\langle S, \cdot, \tau, \tau^*, \kappa\rangle$ has the property of being a $\circ$-algebra, defined as follows:

Definition 2. *A structure $\langle S, \cdot, \tau, \tau^*, \kappa\rangle$, with $\cdot : S^2 \to S$, $\tau, \tau^* : S \to S$, and $\kappa : \mathscr{P}(S) \setminus \{\emptyset\} \to S$, is called a* $\circ$-algebra *if:*

(A1) $(S, \cdot)$ *is a semigroup, namely, for every* $a, b, c \in S$, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$,

(A2) τ *is* compatible to the right*, namely, for every* $a, b \in S$ *and every* $n > 0$, $(a \cdot b)^\tau = a \cdot (b \cdot a)^\tau$ *and* $(a^n)^\tau = a^\tau$,

(A3) τ^* *is* compatible to the left*, namely, for every* $a, b \in S$ *and every* $n > 0$, $(b \cdot a)^{\tau^*} = (a \cdot b)^{\tau^*} \cdot a$ *and* $(a^n)^{\tau^*} = a^{\tau^*}$,

(A4) κ *is* compatible with shuffles*, namely, for every non-empty subset* P *of* S*, every element* c *in* P*, every subset* Q *of* P*, and every non-empty subset* R *of* $\{P^\kappa, a \cdot P^\kappa, P^\kappa \cdot b, a \cdot P^\kappa \cdot b \ : \ a, b \in P\}$*, we have* $P^\kappa = P^\kappa \cdot P^\kappa = P^\kappa \cdot c \cdot P^\kappa = (P^\kappa)^\tau = (P^\kappa \cdot c)^\tau = (P^\kappa)^{\tau^*} = (c \cdot P^\kappa)^{\tau^*} = (Q \cup R)^\kappa$.

The typical $\circ$-algebra is:

Lemma 1. *For every alphabet* A, $\langle A^\circ, \cdot, \omega, \omega^*, \eta\rangle$ *is a* $\circ$*-algebra.*

Proof. By a systematic analysis of Axioms A1-A4. □

Furthermore, as we mentioned above, every $\circ$-semigroup induces a $\circ$-algebra.

Lemma 2. *For every* $\circ$*-semigroup* $\langle S, \pi\rangle$*,* $\langle S, \cdot, \tau, \tau^*, \kappa\rangle$ *is a* $\circ$*-algebra, where the operators* $\cdot, \tau, \tau^*$*, and* κ *are those induced by* π*.*

Extension of $\circ$-algebras to countable products. Here, we aim at proving a converse to Lemma 2, namely, that every *finite* $\circ$-algebra $\langle S, \cdot, \tau, \tau^*, \kappa\rangle$ can be uniquely extended into a unique $\circ$-semigroup $\langle S, \pi\rangle$ (Theorem 1). We assume all words are over the alphabet S. The objective of the construction is to attach to each word u over S a 'value' in S. Furthermore, this value needs to be unique.

The central objects in this proof are *evaluation trees*, i.e., infinite trees describing how a word in S° can be evaluated into an element of S. We begin with condensation trees which are convenient representations for nested condensations. The nodes of a condensation tree are convex subsets of the linear ordering and the descendant relation is the inclusion. The set of children of each node defines a condensation. Furthermore, in order to provide an induction parameter, we require that the branches of a condensation tree are finite (but their length may not be uniformly bounded).

Definition 3. *A* condensation tree *over a linear ordering α is a set T of non-empty convex subsets of α such that:*

- $\alpha \in T$,
- *for all I, J in T, either $I \subseteq J$ or $J \subseteq I$ or $I \cap J = \emptyset$,*
- *for all $I \in T$, the union of all $J \in T$ such that $J \subsetneq I$ is either I or $\emptyset$,*
- *every subset of T totally ordered by inclusion is finite.*

Elements in T are called *nodes*. The node α is called the *root* of the tree. Nodes minimal for $\subseteq$ are called *leaves*. Given $I, J \in T$ such that $I \subsetneq J$ and there exist no $K \in T$ such that $I \subsetneq K \subsetneq J$, then I is called a *child* of J and J the *parent* of I. According to the definition, if I is an *internal node*, i.e., is not a leaf, then it has a set of children $\mathtt{children}_T(I)$ which forms a partition of I. This partition consisting of convexes, it corresponds naturally to a condensation of $\alpha|_I$. When the tree T is clear from the context, we will denote by $\mathtt{children}(I)$ the set of all children of I in T and, by extension, the corresponding condensation and the corresponding condensed linear ordering.

Since the branches of a condensation tree are finite, an ordinal rank, called *foundation rank*, can be associated with such a tree. This is the smallest ordinal β that enables a labeling of the nodes by ordinals less than or equal to β such that the label of each node is strictly greater than the labels of its children. This rank allows us to make proofs by induction (see also [7] for similar definitions).

We now introduce evaluation trees. Intuitively, these are condensation trees where each internal node has an associated value in S that can be 'easily computed' from the values of its children. Here we consider a word u 'easy to compute' if the following function π_0 is defined on u:

Definition 4. *Let π_0 be the partial function from S° to S such that:*

- $\pi_0(ab) = a \cdot b$ *for all $a, b \in S$,*
- $\pi_0(s^\omega) = s^\tau$ *for all $s \in S$,*
- $\pi_0(s^{\omega^*}) = s^{\tau^*}$ *for all $s \in S$,*
- $\pi_0(P^\eta) = P^\kappa$ *for all non-empty sets $P \subseteq S$,*
- *in any other case π_0 is undefined.*

An evaluation tree *over a linear ordering α is a pair $\mathcal{T} = \langle T, \gamma \rangle$ in which T is a condensation tree over α and γ is a function from T to S such that for every internal node $I \in T$, $\gamma(I) = \pi_0(\gamma(\mathtt{children}(I))$, where $\gamma(\mathtt{children}(I))$ denotes the word with domain $\mathtt{children}(I)$ labeling each position $J \in \mathtt{children}(I)$ with $\gamma(J)$ (note that we assume that $\pi_0(\gamma(\mathtt{children}(I))$ is defined). The* value *of $\langle T, \gamma \rangle$ is $\gamma(\alpha)$, i.e., the value of the root.*

An evaluation tree *$\mathcal{T} = \langle T, \gamma \rangle$ over a word u is an evaluation tree over the domain of u such that the leaves of T are singletons and $\gamma(\{x\}) = u(x)$ for all x in the domain of u.*

The next propositions are central in the study of evaluation trees.

Proposition 1. *For every word u, there exists an evaluation tree over u.*

The proof here resembles the construction used by Shelah in his proof of decidability of the monadic second-order theory of orders from [12]. In particular, it uses the theorem of Ramsey [9], as well as a lemma stating that every non-trivial word indexed by a dense linear ordering has a perfect shuffle as a factor. We remark that the above proposition does not use any of the Axioms A1-A4.

Proposition 2. *Two evaluation trees over the same word have the same value.*

The proof of this result is quite involved and it heavily relies on the use of Axioms A1-A4 (each axiom can be seen as an instance of Proposition 2 in some special cases of computation trees of height 2). The proof makes also use of Proposition 1. Note that, as opposed to Proposition 1, Proposition 2 has no counterpart in [12].

Using the above results, the proof of the following result is relatively easy:

Theorem 1. *For every finite $\circ$-algebra $\langle S, \cdot, \tau, \tau^*, \kappa\rangle$, there exists a unique product π that defines $\langle S, \cdot, \tau, \tau^*, \kappa\rangle$.*

Proof. Given a word w with domain α, one defines $\pi(w)$ to be the value of some evaluation tree over w (the evaluation tree exists by Proposition 1 and the value $\pi(w)$ is unique by Proposition 2).

We prove that π is associative. Let $\sim$ be a condensation of the domain α. For all $I \in \alpha/\sim$, let $\mathcal{T}_I$ be some evaluation tree over $w|_I$. Let also $\mathcal{T}'$ be some evaluation tree over the word $w' = \prod_{I\in\alpha/\sim} \pi(w|_I)$. One constructs an evaluation tree $\mathcal{T}$ over w by first lifting $\mathcal{T}'$ from the linear ordering $\alpha/\sim$ to α (this is done by replacing each node J in $\mathcal{T}'$ by $\bigcup J$) and then substituting each leaf of $\mathcal{T}'$ corresponding to some class $I \in \alpha/\sim$ with the subtree $\mathcal{T}_I$. The last step is possible (i.e., respects the definition of evaluation tree) since the value of each evaluation tree $\mathcal{T}_I$ is $\pi(w|_I)$, which coincides with the value $w'(I)$ at the leaf I of $\mathcal{T}'$. By Proposition 2, the resulting evaluation tree $\mathcal{T}$ has the same value as $\mathcal{T}'$ and this witnesses that $\pi(w) = \pi\left(\prod_{I\in\alpha/\sim} \pi(w|_I)\right)$.

What remains to be done is to prove that indeed the above choice of π defines $\cdot, \tau, \tau^*, \kappa$. This requires a case by case analysis. □

Let us conclude with a decidability result.

Theorem 2. *Emptiness of $\circ$-languages recognizable by $\circ$-algebras is decidable.*

Proof (principle of the algorithm). It is sufficient to describe an algorithm which given a set of elements $A \subseteq S$ computes the set $X = \{\pi(u) : u \in A^\circ\}$. For this, one just has to saturate A under the operations $\cdot, \tau, \tau^*, \kappa$ yielding the set $\langle A\rangle$. Indeed, it is easy to prove that $\langle A\rangle \subseteq X$, since this inclusion holds for A and is preserved under each operation of the saturation. The converse inclusion is established using Proposition 1. □

4 From Monadic Second-Order Logic to $\circ$-Algebras

Let us recall that monadic second-order (MSO) logic is the extension of first-order logic with set quantifiers. We assume the reader to have some familiarity

with this logic as well as with the Büchi approach for translating MSO formulas into automata. A good survey can be found in [13]. We refer to the $\forall$-*fragment* as the set of formulas that start with a block of universal set quantifiers, followed by a first-order formula. The $\exists\forall$-*fragment* consists of formulas starting with a block of existential set quantifiers followed by a formula of the $\forall$-fragment.

Here, we mimic Büchi's technique and show a relatively direct consequence of the above results, namely that MSO formulas can be translated to $\circ$-algebras:

Proposition 3. *The MSO definable languages are effectively $\circ$-recognizable.*

Let us remark that we could have equally well used the composition method of Shelah for establishing Proposition 3. Indeed, given an MSO definable language, a $\circ$-algebra recognizing it can be directly extracted from [12].

Our chosen proof for Proposition 3 follows Büchi's approach, namely, we establish sufficiently many closure properties of $\circ$-recognizable language. Then, each construction of the logic can be translated into an operation on languages. To disjunction corresponds union, to conjunction corresponds intersection, to negation corresponds complement, etc. We assume the reader to be familiar with this approach (in particular the coding of the valuations of free variables).

The closure under intersection, union, and complement are, as usual, easy to obtain. The languages corresponding to atomic predicates are also very easily shown to be $\circ$-recognizable. What remains to be proved is the closure under projection. Given a language of $\circ$-words L over some alphabet A, and a mapping h from A to another alphabet B, the *projection of L by h* is simply $h(L)$ (h being extended component-wise to $\circ$-words, and $\circ$-languages). It is classical that this projection operation is what is necessary for obtaining the closure under existential quantification at the logical level. Hence, we just need to prove:

Lemma 3. *The $\circ$-recognizable languages are effectively closed under projections.*

Proof (sketch). We first describe the construction for a given $\circ$-semigroup $\langle S, \pi\rangle$. The projection is obtained, as it is usual, by a powerset construction, i.e., we aim at providing a $\circ$-product over $\mathscr{P}(S)$. Given two words u and U over S and $\mathscr{P}(S)$ respectively, we write $u \in U$ when $\mathcal{D}om(u) = \mathcal{D}om(U)$ and $u(x) \in U(x)$ for all $x \in \mathcal{D}om(U)$. We define the mapping $\widetilde{\pi}$ from $(\mathscr{P}(S))^\circ$ to $\mathscr{P}(S)$ by

$$\widetilde{\pi}(U) \ =^{\text{def}} \ \{\pi(u) \ : \ u \in U\} \qquad \text{for all } U \in (\mathscr{P}(S))^\circ.$$

Let us show that $\widetilde{\pi}$ is associative. Consider a word U over $\mathscr{P}(S)$ and a condensation $\sim$ of its domain. Then,

$$\widetilde{\pi}(U) \ = \ \Big\{\pi(u) \ : \ u \in U\Big\} \ = \ \Big\{\pi\Big(\prod\nolimits_{I\in\alpha/\sim} \pi(u|_I)\Big) \ : \ u \in U\Big\}$$
$$= \ \Big\{\pi\Big(\prod\nolimits_{I\in\alpha/\sim} a_I\Big) \ : \ a_I \in \widetilde{\pi}(U|_I) \text{ for all } I \in \alpha/\sim\Big\} \ = \ \widetilde{\pi}\Big(\prod\nolimits_{I\in\alpha/\sim} \widetilde{\pi}(U|_I)\Big),$$

where the second equality is derived from the associativity of π. Hence $(\mathscr{P}(S), \widetilde{\pi})$ is a $\circ$-semigroup. It is just a matter of writing to show that $\langle \mathscr{P}(S), \widetilde{\pi}\rangle$ recognizes any projection of a language recognized by $\langle S, \pi\rangle$.

Thanks to Lemma 2 and Theorem 1, the above construction can be performed at the level of the $\circ$-algebra $\langle S, \cdot, \tau, \tau^*, \kappa\rangle$, namely, there exists a $\circ$-algebra $\langle \mathscr{P}(S), \widetilde{\cdot}, \widetilde{\tau}, \widetilde{\tau^*}, \widetilde{\kappa}\rangle$ corresponding to $\langle \mathscr{P}(S), \widetilde{\pi}\rangle$. The problem is that this may, a priori, be non-effective. However, using a more careful analysis, it is possible to show the effectiveness of the construction.

Let us give some intuition on how one can compute $P^{\widetilde{\kappa}} = \widetilde{\pi}(P^\eta)$ given a set $P = \{A_1, ..., A_k\}$, with $A_1, ..., A_k \subseteq S$ and $k \geq 1$. This is the most difficult operator. Since $P^{\widetilde{\kappa}} = \{\pi(u) \ : \ u \in U,\ u \in P^\eta\}$, this is very similar to computing $\{\pi(u) \ : \ u \in A^\circ\}$ as done in the proof of Theorem 2. One just needs to restrict the set of considered words u to be the ones such that $u \in U$ for some $U \in P^\eta$. This can be achieved by performing a product of S with a $\circ$-algebra which recognizes the single-word language $\{P^\eta\}$, and then applying the saturation process of Theorem 2 on the resulting $\circ$-algebra. □

5 From $\circ$-Algebras to Monadic Second-Order Logic

We have seen in the previous section that every MSO formula defines a $\circ$-recognizable language. In this section, we sketch the proof of the converse.

Theorem 3. *The $\circ$-recognizable languages are effectively MSO definable. Furthermore, such languages are definable in the $\exists\forall$-fragment of MSO logic.*

We fix for the remaining of the section a morphism h from $\langle A^\circ, \prod\rangle$ to a $\circ$-semigroup $\langle S, \pi\rangle$, with S finite. Let F be some subset of S. Let also $\cdot, \tau, \tau^*, \kappa$ be defined from π. Our goal is to show that $L = h^{-1}(F)$ is MSO definable. It is sufficient for this to show that for every $s \in S$, the language

$$\pi^{-1}(s) \ = \ \{w \in S^\circ \ : \ \pi(w) = s\} \ ,$$

is defined by some MSO formula φ_s. This establishes that $L = \bigcup_{s\in F} h^{-1}(s)$ is defined by the disjunction $\bigvee_{s\in F} \varphi'_s$, where φ'_s is obtained from φ_s by replacing every occurrence of an atom $t(x)$, with $t \in S$, by $\bigvee_{a\in h^{-1}(t)\cap A} a(x)$.

A good approach for defining $\pi^{-1}(s)$ is to use a formula which, given $w \in S^\circ$, guesses some object 'witnessing' $\pi(w) = s$. The only objects that we have seen so far and that are able to 'witness' $\pi(w) = s$ are evaluation trees. Unfortunately, there is no way an MSO formula can guess an evaluation tree, since their height cannot be uniformly bounded. That is why we use another kind of object for witnessing $\pi(w) = a$: the so-called Ramsey split, introduced just below.

Ramsey splits. Ramsey splits are not directly applied to words, but to additive labellings. An *additive labeling* σ from a linear ordering α to a semigroup $\langle S, \cdot\rangle$ (in particular, this will be a $\circ$-semigroup in our case) is a function that maps any pair of elements $x < y$ from α to an element $\sigma(x, y) \in S$ in such a way that $\sigma(x, y) \cdot \sigma(y, z) = \sigma(x, z)$ for all $x < y < z$ in α.

Given two positions $x < y$ in a word w, denote by $[x, y)$ the interval $\{z \ : \ x \leq z < y\}$. Given a word w and two positions $x < y$ in it, we define $\sigma_w(x, y) \in S$ to be $\pi(w|_{[x,y)})$. We just mention σ whenever w is clear from the context. Quite

naturally, σ_w is additive since for all $x < y < z$, we have $\sigma(x,y) \cdot \sigma(y,z) = \pi\big(w|_{[x,y)}\big) \cdot \pi\big(w|_{[y,z)}\big) = \pi\big(w|_{[x,y)} w|_{[y,z)}\big) = \pi\big(w|_{[x,z)}\big) = \sigma(x,z)$.

Definition 5. *A* split *of height n of a linear ordering α is a function $g : \alpha \to [1,n]$. Two elements $x, y \in \alpha$ are called* (k-)neighbors *iff $g(x) = g(y) = k$ and $g(z) \leq k$ for all $z \in [x,y] \cup [y,x]$ (note that neighborhood is an equivalence). The split g is called* Ramsey for some additive labeling σ *iff for all equivalence classes $X \subseteq \alpha$ for the neighborhood relation, there is an idempotent $e \in S$ such that $\sigma(x,y) = e$ for all $x < y$ in X.*

Theorem 4 (Colcombet [5]). *For every finite semigroup $\langle S, \cdot \rangle$, every linear ordering α, and every additive labeling σ from α to $\langle S, \cdot \rangle$, there is a split of α which is Ramsey for σ and which has height at most $2|S|$.*

From $\circ$-recognizable to MSO definable. The principle is to construct a formula which, given a word w, guesses a split of height at most $2|S|$, and uses it for representing the function which to every convex set I associates $\pi(w|_I)$. For the explanations, we assume that some word w is fixed, that its domain is α, and that σ is the additive labeling over α derived from w. We remark, however, that all constructions are uniform and do not depend on w.

We aim at constructing a formula $\texttt{evaluate}_s$, for each $s \in S$, which holds over a word w iff $\pi(w) = s$. The starting point is to guess:

- a split g of α of height at most $2|S|$, and;
- a function e mapping each position $x \in \alpha$ to an idempotent of S.

The intention is that a choice of g, e by the formula is good when the split g is Ramsey for σ and the function e maps each x to the idempotent $e(x)$ that arises when the neighborhood class of x is considered in the definition of Ramseyness. In such a case, we say that (g,e) is *Ramsey*. Observe that neither g nor e can be represented by a single monadic variable. However, since both g and e are functions from α to sets of bounded size ($2|S|$ for g, and $|S|$ for e), one can guess them using a fixed number of monadic variables. This kind of coding is standard, and from now on we shall use explicitly the mappings g and e in MSO formulas.

Knowing a Ramsey pair (g,e) is an advance toward computing the value of a word. Indeed, Ramsey splits can be used as 'accelerating structures' in the sense that every computation of some $\pi(w|_I)$ for a convex subset I becomes significantly easier when a Ramsey split is known, namely, first-order definable. This is formalized by the following lemma.

Lemma 4. *For all $s \in S$, there is a first-order formula $\texttt{value}_s(g,e,X)$, such that for every convex subset I:*

- *if (g,e) is Ramsey, then $\texttt{value}_s(g,e,I)$ holds iff $\pi(w|_I) = s$,*
- *if both $\texttt{value}_s(g,e,I)$ and $\texttt{value}_t(g,e,I)$ hold, then $s = t$.*

One sees those formulas as defining a partial function $\texttt{value}$ mapping g, e, I to some element $s \in S$ (the second item enforces that there is no ambiguity about the value, namely, that this is a function and not a relation). From now we simply use the notation $\texttt{value}(g,e,I)$ as if it were a function.

One needs now to enforce that $\texttt{value}(g, e, I)$ coincides with $\pi(w|_I)$, even without assuming that (g, e) is Ramsey. For this, one uses condensations. A priori, a condensation is not representable by monadic variables, since it is a binary relation. However, any set $X \subseteq \alpha$ naturally defines the relation $\approx_X$ such that $x \approx_X y$ iff either $[x, y] \subseteq X$, or $[x, y] \cap X = \emptyset$. It is easy to check that this relation is a condensation. A form of converse result also holds:

Lemma 5. *For all condensations $\sim$, there is X such that $\sim$ and $\approx_X$ coincide.*

Lemma 5 tells us that it is possible to work with condensations as if they were monadic variables. In particular, we use condensation variables in the sequel, which in fact are implemented by the set obtained from Lemma 5.

Given a convex subset I of α and some condensation $\sim$ of $\alpha|_I$, we denote by $w[I, \sim]$ the word with domain $\beta = (\alpha|_I)/\sim$ in which every $\sim$-equivalence class J is labeled by $\texttt{value}(g, e, J)$. One defines the formula $\texttt{consistency}(g, e)$ which checks that for all convex subsets I and all condensations $\sim$ of $\alpha|_I$ (thanks to Lemma 5), the following conditions hold:

(C1) if I is a singleton $\{x\}$, then $\texttt{value}(g, e, I) = w(x)$,

(C2) if $w[I, \sim] = st$ for some $s, t \in S$, then $\texttt{value}(g, e, I) = s \cdot t$,

(C3) if $w[I, \sim] = s^\omega$ for some $s \in S$, then $\texttt{value}(g, e, I) = s^\tau$,

(C4) if $w[I, \sim] = s^{\omega^*}$ for some $s \in S$, then $\texttt{value}(g, e, I) = s^{\tau^*}$,

(C5) if $w[I, \sim] = P^\eta$ for some $P \subseteq S$, then $\texttt{value}(g, e, I) = P^\kappa$.

For some fixed I and $\sim$, the above tests require access to the elements $w[I, \sim](J)$, where J is a $\sim$-equivalence class of $\alpha|_I$. Since $\sim$-equivalence of two positions $x, y \in \alpha|_I$ is first-order definable, we know that for every position $x \in \alpha|_I$, the element $\texttt{value}(g, e, [x]_\sim)$ is first-order definable from x. This shows that the above properties can be expressed by first-order formulas and hence $\texttt{consistency}(g, e)$ is in the $\forall$-fragment.

The last key argument is to propagate the 'local consistency' constraints C1–C5 to a 'global consistency' property. This is done by the following lemma.

Lemma 6. *If $\texttt{consistency}(g, e)$ holds, then $\texttt{value}(g, e, I) = \pi(w|_I)$ for all convex subsets I of α.*

This lemma implies Theorem 3. We claim indeed that, given $s \in S$, the language $\pi^{-1}(s)$ is defined by the following formula in the $\exists\forall$-fragment of MSO:

$$\texttt{evaluate}_s \ =^{\text{def}} \ \exists g. \exists e.\ \texttt{consistency}(g, e) \wedge \texttt{value}(g, e, \alpha) = s \ .$$

Let $\pi(w) = s$. One can find a Ramsey pair (g, e) using Theorem 4. Lemma 4 then implies $\pi(w|_I) = \texttt{value}(g, e, I)$ for all convex subsets I. Since π is a product, the constraints C1–C5 are satisfied and $\texttt{consistency}(g, e)$ holds. This proves that $\texttt{evaluate}_s$ holds. Conversely, if $\texttt{evaluate}_s$ holds, then $\texttt{consistency}(g, e)$ holds for some (g, e). Lemma 6 then implies $\pi(w) = \pi(w|_\alpha) = \texttt{value}(g, e, \alpha) = s$. □

6 Conclusion

We have introduced an algebraic notion of recognizability for languages of countable words and we have shown the correspondence with the family of languages definable in MSO logic. As a byproduct of this result, it follows that MSO logic interpreted over countable words collapses to its $\exists\forall$-fragment (hence, since it is closed under complementation, it also collapses to its $\forall\exists$-fragment). This collapse result is optimal, in the sense that there exist definable languages that are not definable in the $\exists$-fragment, nor in the $\forall$-fragment. An example of such a language is the set of all scattered words over $\{a\}$ and all non-scattered words over $\{b\}$: checking that a word is scattered requires a universal quantification over the sub-orderings of its domain and, conversely, checking that a word is not scattered requires an existential quantification.

Acknowledgments. We are grateful to Achim Blumensath and the anonymous referees for their numerous comments on this work.

References

[1] Bedon, N., Bès, A., Carton, O., Rispal, C.: Logic and rational languages of words indexed by linear orderings. Theoretical Computer Science 46(4), 737–760 (2010)
[2] Bruyère, V., Carton, O.: Automata on linear orderings. Journal of Computer and System Sciences 73(1), 1–24 (2007)
[3] Büchi, J.R.: Transfinite automata recursions and weak second order theory of ordinals. In: Proc. Int. Congress Logic, Methodology, and Philosophy of Science, Jerusalem, pp. 2–23. North Holland, Amsterdam (1964)
[4] Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proceedings of the International Congress for Logic, Methodology and Philosophy of Science, pp. 1–11. Stanford University Press, Stanford (1962)
[5] Colcombet, T.: Factorisation forests for infinite words and applications to countable scattered linear orderings. Theoretical Computer Science 411, 751–764 (2010)
[6] Feferman, S., Vaught, R.: The first-order properties of products of algebraic systems. Fundamenta Mathematicae 47, 57–103 (1959)
[7] Kechris, A.S.: Classical Descriptive Set Theory. Graduate Texts in Mathematics. Springer, Heidelberg (1995)
[8] Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Transactions of the American Mathematical Society 141, 1–35 (1969)
[9] Ramsey, F.P.: On a problem of formal logic. Proceedings of the London Mathematical Society 30, 264–286 (1929)
[10] Rispal, C., Carton, O.: Complementation of rational sets on countable scattered linear orderings. International Journal of Foundations of Computer Science 16(4), 767–786 (2005)
[11] Rosenstein, J.G.: Linear Orderings. Academic Press, London (1982)
[12] Shelah, S.: The monadic theory of order. Annals of Mathematics 102, 379–419 (1975)
[13] Thomas, W.: Languages, automata, and logic. In: Handbook of Formal Languages, vol. 3, pp. 389–455. Springer, Heidelberg (1997)

Algebraic Independence and Blackbox Identity Testing*

Malte Beecken, Johannes Mittmann, and Nitin Saxena

Hausdorff Center for Mathematics, Bonn, Germany
{malte.beecken,johannes.mittmann,nitin.saxena}@hcm.uni-bonn.de

Abstract. Algebraic independence is an advanced notion in commutative algebra that generalizes independence of linear polynomials to higher degree. The transcendence degree (trdeg) of a set $\{f_1, \ldots, f_m\} \subset \mathbb{F}[x_1, \ldots, x_n]$ of polynomials is the maximal size r of an algebraically independent subset. In this paper we design blackbox and efficient linear maps φ that reduce the number of variables from n to r but maintain $\operatorname{trdeg}\{\varphi(f_i)\}_i = r$, assuming f_i's sparse and small r. We apply these fundamental maps to solve several cases of blackbox identity testing:

1. Given a circuit C and sparse subcircuits $f_1, \ldots, f_m$ of trdeg r such that $D := C(f_1, \ldots, f_m)$ has polynomial degree, we can test blackbox D for zeroness in $\operatorname{poly}(\operatorname{size}(D))^r$ time.
2. Define a $\Sigma\Pi\Sigma\Pi_\delta(k, s, n)$ circuit C to be of the form $\sum_{i=1}^{k} \prod_{j=1}^{s} f_{i,j}$, where $f_{i,j}$ are sparse n-variate polynomials of degree at most δ. For $k = 2$, we give a $\operatorname{poly}(\delta s n)^{\delta^2}$ time blackbox identity test.
3. For a general depth-4 circuit we define a notion of rank. Assuming there is a rank bound R for minimal simple $\Sigma\Pi\Sigma\Pi_\delta(k, s, n)$ identities, we give a $\operatorname{poly}(\delta s n R)^{Rk\delta^2}$ time blackbox identity test for $\Sigma\Pi\Sigma\Pi_\delta(k, s, n)$ circuits. This partially generalizes the state of the art of depth-3 to depth-4 circuits.

The notion of trdeg works best with large or zero characteristic, but we also give versions of our results for arbitrary fields.

1 Introduction

Polynomial identity testing (PIT) is the problem of checking whether a given n-variate arithmetic circuit computes the zero polynomial in $\mathbb{F}[x_1, \ldots, x_n]$. It is a central question in complexity theory as circuits model computation and PIT leads us to a better understanding of circuits. There are several classical randomized algorithms known [9,28,30,8,19,4] that solve PIT. The basic Schwartz-Zippel test is: Given a circuit $C(x_1, \ldots, x_n)$, check $C(\overline{a}) = 0$ for a random $\overline{a} \in \overline{\mathbb{F}}^n$. Finding a deterministic polynomial time test, however, has been more difficult and is currently open. Derandomization of PIT is well motivated by a host of algorithmic applications, eg. bipartite matching [20] and matrix completion [21], and

* The first two authors are grateful to the Bonn International Graduate School in Mathematics for research funding.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 137–148, 2011.

connections to sought-after super-polynomial lower bounds [13,14]. Especially, *blackbox* PIT (i.e. circuit C is given as a blackbox and we could only make oracle queries) has direct connections to lower bounds for the permanent [2,3]. Clearly, finding a blackbox PIT test for a family of circuits $\mathcal{F}$ boils down to efficiently designing a *hitting set* $\mathcal{H} \subset \overline{\mathbb{F}}^n$ such that: Given a nonzero $C \in \mathcal{F}$, there exists an $\overline{a} \in \mathcal{H}$ that *hits* C, i.e. $C(\overline{a}) \neq 0$.

The attempts to solve blackbox PIT have focused on restricted circuit families. A natural restriction is *constant depth*. Agrawal & Vinay [5] showed that a blackbox PIT algorithm for depth-4 circuits would (almost) solve PIT for general circuits (and prove exponential circuit lower bounds for permanent). The currently known blackbox PIT algorithms work only for further restricted depth-3 and depth-4 circuits. The case of *bounded top fanin* depth-3 circuits has received great attention and has blackbox PIT algorithms [27]. The analogous case for depth-4 circuits is open. However, with the additional restriction of *multilinearity* on all the multiplication gates, there is a blackbox PIT algorithm [24]. The latter is somewhat subsumed by the PIT algorithms for constant-read multilinear formulas [6]. To save space we would not go into the rich history of PIT and instead refer to the survey [29].

A recurring theme in the blackbox PIT research on depth-3 circuits has been that of *rank*. If we consider a $\Sigma\Pi\Sigma(k,d,n)$ circuit $C = \sum_{i=1}^{k} \prod_{j=1}^{d} \ell_{i,j}$, where $\ell_{i,j}$ are linear forms in $\mathbb{F}[x_1,\ldots,x_n]$, then $\mathrm{rk}(C)$ is defined to be the linear rank of the set of forms $\{\ell_{i,j}\}_{i,j}$ each viewed as a vector in $\mathbb{F}^n$. This raises the natural question: Is there a generalized notion of rank for depth-4 circuits as well, and more importantly, one that is useful in blackbox PIT? We answer this question affirmatively in this paper. Our notion of rank is via *transcendence degree* (short, trdeg), which is a basic notion in commutative algebra. To show that this notion applies to PIT requires relatively advanced algebra and new tools that we build.

Consider polynomials $\{f_1,\ldots,f_m\}$ in $\mathbb{F}[x_1,\ldots,x_n]$. They are called *algebraically independent* (over $\mathbb{F}$) if there is no nonzero polynomial $F \in \mathbb{F}[y_1,\ldots,y_m]$ such that $F(f_1,\ldots,f_m) = 0$. When those polynomials are *algebraically dependent* then such an F exists and is called an *annihilating polynomial* of $f_1,\ldots,f_m$. The *transcendence degree*, $\mathrm{trdeg}\{f_1,\ldots,f_m\}$, is the maximal number r of algebraically independent polynomials in the set $\{f_1,\ldots,f_m\}$. Though intuitive, it is nontrivial to prove that r is at most n.

The notion of trdeg has appeared in complexity theory in several contexts. Kalorkoti [15] used it to prove an $\Omega(n^3)$ formula size lower bound for $n \times n$ determinant. In the works [10,11] studying the *entropy* of polynomial mappings $(f_1,\ldots,f_m) : \mathbb{F}^n \to \mathbb{F}^m$, trdeg is a natural measure of entropy when the field has large or zero characteristic. Finally, the complexity of the annihilating polynomial is studied in [17]. However, our work is the first to study trdeg in the context of PIT.

1.1 Our Main Results

Our first result shows that a general arithmetic circuit is sensitive to the trdeg of its input.

Theorem 1. *Let C be an m-variate circuit. Let $f_1, \ldots, f_m$ be ℓ-sparse, δ-degree, n-variate polynomials with trdeg r. Suppose we have oracle access to the n-variate d-degree circuit $C' := C(f_1, \ldots, f_m)$. There is a blackbox* $\mathrm{poly}(\mathrm{size}(C') \cdot d\ell\delta)^r$ *time test to check $C' = 0$ (assuming a zero or larger than δ^r characteristic).*

We also give an algorithm that works for all fields but has a worse time complexity. Note that the above theorem seems nontrivial even for a constant m, say $C' = C(f_1, f_2, f_3)$, as the output of C' may not be sparse and f_i's are of arbitrary degree and arity. In such a case r is constant too and the theorem gives a polynomial time test. Another example, where r is constant but both m and n are variable, is: $f_i := (x_1^i + x_2^2 + \cdots + x_n^2)x_n^i$ for $i \in [m]$. (Hint: $r \leq 3$.)

Our next two main results concern depth-4 circuits. We use the notation $\Sigma\Pi\Sigma\Pi_\delta(k, s, n)$ to denote circuits (over a field $\mathbb{F}$) of the form,

$$C := \sum_{i=1}^{k} \prod_{j=1}^{s} f_{i,j} \tag{1}$$

where $f_{i,j}$'s are sparse n-variate polynomials of maximal degree δ. Note that when $\delta = 1$ this notation agrees with that of a $\Sigma\Pi\Sigma$ circuit. Currently, the PIT methods are not even strong enough to study $\Sigma\Pi\Sigma\Pi_\delta(k, s, n)$ circuits with both *top* fanin k and *bottom* fanin δ *bounded*. It is in this spectrum that we make exciting progress.

Theorem 2. *Let C be a $\Sigma\Pi\Sigma\Pi_\delta(2, s, n)$ circuit over an arbitrary field. There is a blackbox* $\mathrm{poly}(\delta sn)^{\delta^2}$ *time test to check $C = 0$.*

Finally, we define a notion of rank for depth-4 circuits and show its usefulness. For a circuit C, as in (1), we define its *rank*, $\mathrm{rk}(C) := \mathrm{trdeg}\{f_{i,j} \mid i \in [k], j \in [s]\}$. Define $T_i := \prod_{j=1}^{s} f_{i,j}$, for all $i \in [k]$, to be the *multiplication terms* of C. We call C *simple* if $\{T_i | i \in [k]\}$ are coprime polynomials. We call C *minimal* if there is no $I \subsetneq [k]$ such that $\sum_{i \in I} T_i = 0$. Define $R_\delta(k, s)$ to be the smallest r such that: Any $\Sigma\Pi\Sigma\Pi_\delta(k, s, n)$ circuit C that is simple, minimal and zero has $\mathrm{rk}(C) < r$.

Theorem 3. *Let $r := R_\delta(k, s)$ and the characteristic be zero or larger than δ^r. There is a blackbox* $\mathrm{poly}(\delta rsn)^{rk\delta^2}$ *time identity test for $\Sigma\Pi\Sigma\Pi_\delta(k, s, n)$ circuits.*

We give a lower bound of $\Omega(\delta k \log s)$ on $R_\delta(k, s)$ and conjecture an upper bound (better than the trivial ks).

1.2 Organization and Our Approach

A priori it is not clear whether the problem of deciding algebraic independence of given polynomials $\{f_1, \ldots, f_m\}$, over a field $\mathbb{F}$, is even computable. Perron [22] proved that for $m = (n + 1)$ and any field, the annihilating polynomial has degree only exponential in n. We generalize this to any m in Sect. 2.1, hence, deciding algebraic independence (over any field) is computable (alternatively, Gröbner bases can be used). When the characteristic is zero or large, there is

a more efficient criterion due to Jacobi (Sect. 2.2). For using trdeg in PIT we would need to relate it to the *Krull dimension* of algebras (Sect. 2.3).

The central concept that we develop is that of a *faithful homomorphism*. This is a linear map φ from $R := \mathbb{F}[x_1, \ldots, x_n]$ to $\mathbb{F}[z_1, \ldots, z_r]$ such that for polynomials $f_1, \ldots, f_m \in R$ of trdeg r, the images $\varphi(f_1), \ldots, \varphi(f_m)$ are also of trdeg r. Additionally, to be useful, φ should be constructible in a blackbox and efficient way. We give such constructions in Sects. 3.1 and 3.2. The proofs here use Perron's and Jacobi's criterion, but require new techniques as well. The reason why such a φ is useful in PIT is because it preserves the nonzeroness of the circuit $C(f_1, \ldots, f_m)$ (Theorem 11). We prove this by an elegant application of Krull's *principal ideal theorem*.

Once the fundamental machinery is set up, we prove Theorem 1 in Sect. 4 by designing a hitting set using the basic Schwartz-Zippel lemma.

Finally, we apply the faithful homomorphisms to depth-4 circuits. The proof of Theorem 2 is provided in Sect. 5.2 by showing that the maps also preserve gcd's. The rank-based hitting set is constructed in Sect. 5.3 proving Theorem 3.

Due to space constraints most of the proofs are omitted. They can be found in the full version of this paper [7].

2 Preliminaries: Perron, Jacobi and Krull

Let $n \in \mathbb{Z}^+$ and let K be a field of characteristic $\mathrm{ch}(K)$. Throughout this paper, $K[\boldsymbol{x}] = K[x_1, \ldots, x_n]$ is a polynomial ring in n variables over K. $\overline{K}$ denotes the *algebraic closure* of the field. We denote the multiplicative *group of units* of an algebra A by A^*. We use the notation $[n] := \{1, \ldots, n\}$. For $0 \leq r \leq n$, $\binom{[n]}{r}$ denotes the set of r-subsets of $[n]$.

2.1 Perron's Criterion (Arbitrary Characteristic)

An effective criterion for algebraic independence can be obtained by a degree bound for annihilating polynomials. The following theorem provides such a bound for the case of $n+1$ polynomials in n variables.

Theorem 4 (Perron's theorem [23, Thm. 1.1]). *Let $f_i \in K[\boldsymbol{x}]$ be a polynomial of degree $\delta_i \geq 1$, for $i \in [n+1]$. Then there exists a non-zero polynomial $F \in K[y_1, \ldots, y_{n+1}]$ such that $F(f_1, \ldots, f_{n+1}) = 0$ and $\deg(F) \leq (\prod_i \delta_i)/\min_i\{\delta_i\}$.*

In the following corollary we give a degree bound in the general situation, where more variables than polynomials are allowed. Moreover, the bound is in terms of the trdeg of the polynomials instead of the number of variables. We hereby improve [17, Theorem 11] and generalize it to arbitrary characteristic.

Corollary 5 (Degree bound for annihilating polynomials). *Let $f_1, \ldots, f_m \in K[\boldsymbol{x}]$ be algebraically dependent polynomials of maximal degree δ and* trdeg r. *Then there exists a non-zero polynomial $F \in K[y_1, \ldots, y_m]$ of degree at most δ^r such that $F(f_1, \ldots, f_m) = 0$.*

2.2 Jacobi's Criterion (Large or Zero Characteristic)

In large or zero characteristic, the well-known Jacobian criterion yields a more efficient criterion for algebraic independence. The case of large characteristic was dealt with in [10]. By virtue of Theorem 4 our proof could tolerate a slightly smaller characteristic.

Theorem 6 (Jacobian criterion). *Let* $f_1, \ldots, f_m \in K[\boldsymbol{x}]$ *be polynomials of degree at most* δ *and* trdeg r. *Assume that* $\mathrm{ch}(K) = 0$ *or* $\mathrm{ch}(K) > \delta^r$. *Then* $\mathrm{rk}_L(\partial_{x_j} f_i)_{i,j} = r$, *where* $L = K(\boldsymbol{x})$.

2.3 Krull Dimension of Affine Algebras

In this section, we want to highlight the connection between transcendence degree and the Krull dimension of affine algebras. This will enable us to use Krull's principal ideal theorem which is stated below.

In this paper, a *K-algebra* A is always a commutative ring containing K as a subring. The most important example of a K-algebra is $K[\boldsymbol{x}]$. Let A, B be K-algebras. A map $A \to B$ is called a *K-algebra homomorphism* if it is a ring homomorphism that fixes K element-wise.

We want to extend the definition of algebraic independence to algebras. Let $a_1, \ldots, a_m \in A$ and consider the K-algebra homomorphism $\rho : K[\boldsymbol{y}] \to A$, $F \mapsto F(a_1, \ldots, a_m)$, where $K[\boldsymbol{y}] = K[y_1, \ldots, y_m]$. If $\ker(\rho) = \{0\}$, then $\{a_1, \ldots, a_m\}$ is called algebraically independent over K. If $\ker(\rho) \neq \{0\}$, then $\{a_1, \ldots, a_m\}$ is called algebraically dependent over K. For a subset $S \subseteq A$, we define $\mathrm{trdeg}_K S$ as the supremum of $|T|$ over all finite and algebraically independent $T \subseteq S$. The image of $K[\boldsymbol{y}]$ under ρ is the subalgebra of A generated by $a_1, \ldots, a_m$ and is denoted by $K[a_1, \ldots, a_m]$. An algebra of this form is called an *affine K-algebra*, and it is called an *affine K-domain* if it is an integral domain.

The *Krull dimension* of A, denoted by $\dim(A)$, is defined as the supremum over all $r \geq 0$ for which there is a chain $\mathfrak{p}_0 \subsetneq \mathfrak{p}_1 \subsetneq \cdots \subsetneq \mathfrak{p}_r$ of prime ideals $\mathfrak{p}_i \subset A$. It measures how far A is from a field.

Theorem 7 (Dimension and trdeg [18, Prop. 5.10]). *Let* $A = K[a_1, \ldots, a_m]$ *be an affine K-algebra. Then* $\dim(A) = \mathrm{trdeg}_K A = \mathrm{trdeg}_K\{a_1, \ldots, a_m\}$.

Corollary 8. *Let* A, B *be K-algebras and let* $\varphi : A \to B$ *be a K-algebra homomorphism. If A is an affine algebra, then so is* $\varphi(A)$ *and we have* $\dim(\varphi(A)) \leq \dim(A)$. *If, in addition,* φ *is injective, then* $\dim(\varphi(A)) = \dim(A)$.

Theorem 9 (Krull's Hauptidealsatz [12, Cor. 13.11]). *Let A be an affine K-domain and let* $a \in A \setminus (A^* \cup \{0\})$. *Then* $\dim(A/\langle a \rangle) = \dim(A) - 1$.

3 Faithful Homomorphisms: Reducing the Variables

Let $f_1, \ldots, f_m \in K[\boldsymbol{x}]$ be polynomials and let $r := \mathrm{trdeg}\{f_1, \ldots, f_m\}$. Intuitively, r variables should suffice to define $f_1, \ldots, f_m$ without changing their algebraic

relations. So let $K[\boldsymbol{z}] = K[z_1, \ldots, z_r]$ be a polynomial ring with $1 \leq r \leq n$. We want to find a homomorphism $K[\boldsymbol{x}] \to K[\boldsymbol{z}]$ that preserves the transcendence degree of $f_1, \ldots, f_m$. First we give this property a name.

Definition 10. Let $\varphi : K[\boldsymbol{x}] \to K[\boldsymbol{z}]$ be a K-algebra homomorphism. We say φ is *faithful to* $\{f_1, \ldots, f_m\}$ if $\operatorname{trdeg}\{\varphi(f_1), \ldots, \varphi(f_m)\} = \operatorname{trdeg}\{f_1, \ldots, f_m\}$.

The following theorem shows that a faithful homomorphism φ is useful for us. In particular, for a circuit C, we have $C(f_1, \ldots, f_m) = 0$ if and only if $\varphi(C(f_1, \ldots, f_m)) = 0$.

Theorem 11 (Faithful is useful). *Let $A = K[f_1, \ldots, f_m] \subseteq K[\boldsymbol{x}]$. Then φ is faithful to $\{f_1, \ldots, f_m\}$ if and only if $\varphi|_A : A \to K[\boldsymbol{z}]$ is injective.*

Proof. We denote $\varphi_A = \varphi|_A$ and $r = \operatorname{trdeg}\{f_1, \ldots, f_m\}$. If φ_A is injective, then $r = \dim(A) = \dim(\varphi_A(A)) = \operatorname{trdeg}\{\varphi(f_1), \ldots, \varphi(f_m)\}$ by Theorem 7 and Corollary 8. Thus φ is faithful to $\{f_1, \ldots, f_m\}$.

Conversely, let φ be faithful to $\{f_1, \ldots, f_m\}$. Then $\dim(\varphi_A(A)) = r$. Now assume for the sake of contradiction that φ_A is not injective. Then there exists an $f \in A \setminus \{0\}$ such that $\varphi_A(f) = 0$. We have $f \notin K$, because φ fixes K elementwise, and hence $f \notin A^* \cup \{0\}$. Since A is an affine domain, Theorem 9 implies $\dim(A/\langle f \rangle) = r - 1$. Since $f \in \ker(\varphi_A)$, the K-algebra homomorphism $\overline{\varphi}_A : A/\langle f \rangle \to K[\boldsymbol{z}]$, $a + \langle f \rangle \mapsto \varphi_A(a)$ is well-defined and φ_A factors as $\varphi_A = \overline{\varphi}_A \circ \eta$, where $\eta : A \to A/\langle f \rangle$ is the canonical surjection. But then Corollary 8 implies

$$r = \dim(\varphi_A(A)) = \dim(\overline{\varphi}_A(\eta(A))) \leq \dim(\eta(A)) = \dim(A/\langle f \rangle) = r - 1 \ ,$$

a contradiction. It follows that φ_A is injective. □

3.1 A Kronecker-Inspired Map (Arbitrary Characteristic)

The following lemma shows that even *linear* faithful homomorphisms exist for all subsets of polynomials (provided K is large enough, for eg. move to $\overline{K}$ or a large enough field extension [1]). It is a generalization of [17, Claim 11.1] to arbitrary characteristic.

Lemma 12 (Existence). *Let K be an infinite field and let $f_1, \ldots, f_m \in K[\boldsymbol{x}]$ be polynomials of* trdeg r. *Then there exists a linear K-algebra homomorphism $\varphi : K[\boldsymbol{x}] \to K[\boldsymbol{z}]$ which is faithful to $\{f_1, \ldots, f_m\}$.*

Below we want to make this lemma effective. This will be accomplished by substituting constants for all but r of the variables $x_1, \ldots, x_n$. We define a parametrized homomorphism Φ in three steps. First, we decide which variables we want to keep and map them to $z_1, \ldots, z_r$. To the remaining variables we apply a *Kronecker substitution* using a new variable t, i.e. we map the i-th variable to t^{D^i} (for a large D). In the second step, the exponents of t will be reduced modulo some number. Finally, a constant will be substituted for t.

Let $I = \{j_1, \ldots, j_r\} \in \binom{[n]}{r}$ be an index set and let $[n] \setminus I = \{j_{r+1}, \ldots, j_n\}$ be its complement such that $j_1 < \cdots < j_r$ and $j_{r+1} < \cdots < j_n$. Let $D \geq 2$ and define the K-algebra homomorphism

$$\Phi_{I,D} : K[\boldsymbol{x}] \to K[t, \boldsymbol{z}], \qquad x_{j_i} \mapsto \begin{cases} z_i, & \text{for } i = 1, \ldots, r, \\ t^{D^{i-r}}, & \text{for } i = r+1, \ldots, n \ . \end{cases}$$

Now let $p \geq 1$. For an integer $a \in \mathbb{Z}$, we denote by $\lfloor a \rfloor_p$ the integer $b \in \mathbb{Z}$ satisfying $0 \leq b < p$ and $a = b \pmod p$. We define the K-algebra homomorphism

$$\Phi_{I,D,p} : K[\boldsymbol{x}] \to K[t, \boldsymbol{z}], \qquad x_{j_i} \mapsto \begin{cases} z_i, & \text{for } i = 1, \ldots, r, \\ t^{\lfloor D^{i-r} \rfloor_p}, & \text{for } i = r+1, \ldots, n \ . \end{cases}$$

Note that, for $f \in K[\boldsymbol{x}]$, $\Phi_{I,D,p}(f)$ is a representative of the residue class $\Phi_{I,D}(f) \pmod{\langle t^p - 1 \rangle_{K[t,\boldsymbol{z}]}}$. Finally let $c \in \overline{K}$ and define the $\overline{K}$-algebra homomorphism $\Phi_{I,D,p,c} : \overline{K}[\boldsymbol{x}] \to \overline{K}[\boldsymbol{z}]$, $f \mapsto \big(\Phi_{I,D,p}(f)\big)(c, \boldsymbol{z})$. The following lemma bounds the number of bad choices for the parameters p and c.

Lemma 13 (Φ is faithful). *Let $f_1, \ldots, f_m \in K[\boldsymbol{x}]$ be polynomials of degree at most δ and* trdeg *at most r. Let $D > \delta^{r+1}$. Then there exist an index set $I \in \binom{[n]}{r}$ and a prime $p \leq (n + \delta^r)^{8\delta^{r+1}} (\log_2 D)^2 + 1$ such that any subset of $\overline{K}$ of size $\delta^r rp$ contains c such that $\Phi_{I,D,p,c}$ is faithful to $\{f_1, \ldots, f_m\}$.*

In large or zero characteristic, a more efficient version of this lemma can be given (for the same homomorphism Φ). The reason is that we can work with the Jacobian criterion instead of the degree bound for annihilating polynomials. However, we omit the statement of this result here, because we can give a more holistic construction in that case. This will be presented in the following section.

3.2 A Vandermonde-Inspired Map (Large or Zero Characteristic)

To prove Theorem 3, we will need a homomorphism that is faithful to several sets of polynomials simultaneously. The homomorphism Φ constructed in the previous section does not meet this requirement, because its definition depends on a *fixed* subset of the variables $x_1, \ldots, x_n$. In this section we will devise a construction, that treats the variables $x_1, \ldots, x_n$ in a uniform manner. It is inspired by the *Vandermonde matrix*, i.e. $(t^{ij})_{i,j}$.

We define a parametrized homomorphism Ψ in three steps. Let $K[\boldsymbol{z}] = K[z_0, z_1, \ldots, z_r]$, where $1 \leq r \leq n$. Let $D_1, D_2 \geq 2$ and let $D = (D_1, D_2)$. Define the K-algebra homomorphism

$$\Psi_D : K[\boldsymbol{x}] \to K[t, \boldsymbol{z}], \qquad x_i \mapsto t^{D_1^i} + t^{D_2^i} z_0 + \sum_{j=1}^{r} t^{i(n+1)^j} z_j \ ,$$

where $i = 1, \ldots, n$. This map (linear in the z's) should be thought of as a variable reduction from n to $r + 1$. The coefficients of $z_1, \ldots, z_r$ bear resemblance to a

row of a Vandermonde matrix, while that of z_0 (and the constant coefficient) resembles Kronecker substitution. This definition is carefully tuned so that Ψ finally preserves both the trdeg (proven here) and gcd of polynomials (proven in Sect. 5.2). Next let $p \geq 1$ and define the K-algebra homomorphism

$$\Psi_{D,p} : K[\boldsymbol{x}] \to K[t, \boldsymbol{z}], \qquad x_i \mapsto t^{\lfloor D_1^i \rfloor_p} + t^{\lfloor D_2^i \rfloor_p} z_0 + \sum_{j=1}^{r} t^{\lfloor i(n+1)^j \rfloor_p} z_j \ ,$$

where $i = 1, \ldots, n$. Note that, for $f \in K[\boldsymbol{x}]$, $\Psi_{D,p}(f)$ is a representative of the residue class $\Psi_D(f) \pmod{\langle t^p - 1 \rangle_{K[t,\boldsymbol{z}]}}$. Finally let $c \in \overline{K}$ and define the $\overline{K}$-algebra homomorphism $\Psi_{D,p,c} : \overline{K}[\boldsymbol{x}] \to \overline{K}[\boldsymbol{z}]$, $f \mapsto \big(\Psi_{D,p}(f)\big)(c, \boldsymbol{z})$. The following lemma bounds the number of bad choices for the parameters p and c. The proof uses the Jacobian criterion, therefore the lemma has a restriction on $\mathrm{ch}(K)$.

Lemma 14 (Ψ is faithful). *Let $f_1, \ldots, f_m \in K[\boldsymbol{x}]$ be polynomials of sparsity at most ℓ, degree at most δ and* trdeg *at most r. Assume that $\mathrm{ch}(K) = 0$ or $\mathrm{ch}(K) > \delta^r$. Let $D = (D_1, D_2)$ such that $D_1 \geq \max\{\delta r + 1, (n+1)^{r+1}\}$ and $D_2 \geq 2$. Then there exists a prime $p \leq (2nr\ell)^{2(r+1)} (\log_2 D_1)^2 + 1$ such that any subset of $\overline{K}$ of size $\delta r p$ contains c such that $\Psi_{D,p,c}$ is faithful to $\{f_1, \ldots, f_m\}$.*

By trying larger p and c, we can find a Ψ that is faithful to several subsets of polynomials simultaneously. This is an advantage of Ψ over Φ, in addition to being more efficiently constructible.

4 Circuits with Sparse Inputs of Low Transcendence Degree (Proving Theorem 1)

We can now proceed with the first PIT application of faithful homomorphisms. We consider arithmetic circuits of the form $C(f_1, \ldots, f_m)$, where C is a circuit computing a polynomial in $K[\boldsymbol{y}] = K[y_1, \ldots, y_m]$ and $f_1, \ldots, f_m$ are subcircuits computing polynomials in $K[\boldsymbol{x}]$. Thus, $C(f_1, \ldots, f_m)$ computes a polynomial in the subalgebra $K[f_1, \ldots, f_m]$.

Let $C(f_1, \ldots, f_m)$ be of maximal degree d, and let $f_1, \ldots, f_m$ be of maximal degree δ, maximal sparsity ℓ and maximal transcendence degree r. First, we use the faithful homomorphism Ψ from Sect. 3.2 to transform $C(f_1, \ldots, f_m)$ into an r-variate circuit. Then a hitting set for r-variate degree-d polynomials, given by the classical Schwartz-Zippel lemma, is used. The final hitting set construction is efficient for r constant and ℓ, d polynomial in the input size.

Let $n, d, r, \delta, \ell \geq 1$ and let $K[\boldsymbol{z}] = K[z_0, z_1, \ldots, z_r]$. We introduce the following parameters. Define $D = (D_1, D_2)$ by $D_1 := (2\delta n)^{r+1}$ and $D_2 := 2$, and $p_{\max} := (2nr\ell)^{2(r+1)} \lceil \log_2 D_1 \rceil^2 + 1$. Pick arbitrary $H_1, H_2 \subset \overline{K}$ of sizes $\delta r p_{\max}$ resp. $d+1$. Finally, denote $\Psi^{(i)}_{D,p,c} := \Psi_{D,p,c}(x_i) \in \overline{K}[\boldsymbol{z}]$ for $i = 1, \ldots, n$ and define the subset

$$\mathcal{H}_{d,r,\delta,\ell} = \left\{ \big(\Psi^{(1)}_{D,p,c}(\boldsymbol{a}), \ldots, \Psi^{(n)}_{D,p,c}(\boldsymbol{a})\big) \;\middle|\; p \in [p_{\max}],\, c \in H_1,\, \boldsymbol{a} \in H_2^{r+1} \right\} \subset \overline{K}^n \ .$$

The following theorem shows that, over large or zero characteristic, this is a hitting set for the class of circuits under consideration. A version of this theorem for arbitrary characteristic can be found in [7].

Theorem 15. *Assume that* $\mathrm{ch}(K) = 0$ *or* $\mathrm{ch}(K) > \delta^r$. *Then* $\mathcal{H}_{d,r,\delta,\ell}$ *is a hitting set for the class of degree-*d *circuits with inputs being* ℓ*-sparse, degree-*δ *subcircuits of* trdeg *at most* r. *It can be constructed in* $\mathrm{poly}(dr\delta\ell n)^r$ *time.*

5 Depth-4 Circuits with Bounded Top and Bottom Fanin

The second PIT application of faithful homomorphisms is for $\Sigma\Pi\Sigma\Pi_\delta(k, s, n)$ circuits. Our hitting set construction is efficient when the top fanin k and the bottom fanin δ are both bounded. Except for top fanin 2, our hitting set will be *conditional* in the sense that its efficiency depends on a good rank upper bound for depth-4 identities.

5.1 Gcd, Simple Parts and the Rank Bounds

Let $C = \sum_{i=1}^{k} \prod_{j=1}^{s} f_{i,j}$ be a $\Sigma\Pi\Sigma\Pi_\delta(k, s, n)$ circuit, as defined in Sect. 1.1. Note that the parameters bound the circuit degree, $\deg(C) \leq \delta s$. We define $\mathcal{S}(C) := \{f_{i,j} \mid i \in [k] \text{ and } j \in [s]\}$. It is the set of *sparse polynomials* of C (wlog we assume them all to be nonzero). The following definitions are natural generalizations of the corresponding concepts for depth-3 circuits. Recall $T_i := \prod_j f_{i,j}$, for $i \in [k]$, are the multiplication terms of C. The *gcd part* of C is defined as $\gcd(C) := \gcd(T_1, \ldots, T_k)$ (we fix a unique representative among the associated gcds). The *simple part* of C is defined as $\mathrm{sim}(C) := C/\gcd(C) \in \Sigma\Pi\Sigma\Pi_\delta(k, s, n)$. For a subset $I \subseteq [k]$ we denote $C_I := \sum_{i \in I} T_i$.

Recall that if C is simple then $\gcd(C) = 1$ and if it is minimal then $C_I \neq 0$ for all non-empty $I \subsetneq [k]$. Also, recall that $\mathrm{rk}(C)$ is $\mathrm{trdeg}_K \mathcal{S}(C)$, and that $R_\delta(k, s)$ strictly upper bounds the rank of any minimal and simple $\Sigma\Pi\Sigma\Pi_\delta(k, s, n)$ identity. Clearly, $R_\delta(k, s)$ is at most $|\mathcal{S}(C)| \leq ks$ (note: $\mathcal{S}(C)$ cannot all be independent in an identity). On the other hand, we could prove a lower bound on $R_\delta(k, s)$ by constructing identities.

From the simple and minimal $\Sigma\Pi\Sigma$ identities constructed in [26], we obtain the lower bound $R_1(k, s) = \Omega(k)$ if $\mathrm{ch}(K) = 0$, and $R_1(k, s) = \Omega(k \log_p s)$ if $\mathrm{ch}(K) = p > 0$. These identities can be lifted to $\Sigma\Pi\Sigma\Pi_\delta(k, s, n)$ identities by replacing each variable x_i by a product $x_{i,1} \cdots x_{i,\delta}$ of new variables. These examples demonstrate: $R_\delta(k, s) = \Omega(\delta k)$ if $\mathrm{ch}(K) = 0$, and $R_\delta(k, s) = \Omega(\delta k \log_p s)$ if $\mathrm{ch}(K) = p > 0$. This leads us to the following natural conjecture.

Conjecture 16. *We have* $R_\delta(k, s) = \mathrm{poly}(\delta k)$, *if* $\mathrm{ch}(K) = 0$, *and* $R_\delta(k, s) = \mathrm{poly}(\delta k \log_p s)$, *if* $\mathrm{ch}(K) = p > 0$.

The following lemma is a vast generalization of [16, Theorem 3.4] to depth-4 circuits. It suggests how a bound for $R_\delta(k, s)$ can be used to construct a hitting

set for $\Sigma\Pi\Sigma\Pi_\delta(k,s,n)$ circuits. The φ in the statement below should be thought of as a linear map that reduces the number of variables from n to $R_\delta(k,s)+1$.

Lemma 17 (Rank is useful). *Let C be a $\Sigma\Pi\Sigma\Pi_\delta(k,s,n)$ circuit, let $r := R_\delta(k,s)$ and let $\varphi : K[\boldsymbol{x}] \to K[\boldsymbol{z}] = K[z_0,z_1,\ldots,z_r]$ be a linear K-algebra homomorphism that, for all $I \subseteq [k]$, satisfies $\varphi(\mathrm{sim}(C_I)) = \mathrm{sim}(\varphi(C_I))$ and $\mathrm{rk}(\varphi(\mathrm{sim}(C_I))) \geq \min\{\mathrm{rk}(\mathrm{sim}(C_I)), R_\delta(k,s)\}$. Then $C = 0$ iff $\varphi(C) = 0$.*

5.2 Preserving the Simple Part (Towards Theorem 2)

The following lemma shows that Ψ meets the first condition of Lemma 17. This is also the heart of PIT when $k = 2$. The actual hitting set, though, we provide in the next subsection.

Lemma 18 (Ψ preserves the simple part). *Let C be a $\Sigma\Pi\Sigma\Pi_\delta(k,s,n)$ circuit. Let $D_1 \geq 2\delta^2+1$, let $D_1 \geq D_2 \geq \delta+1$ and let $D = (D_1,D_2)$. Then there exists a prime $p \leq (2ksn\delta^2)^{8\delta^2+2}(\log_2 D_1)^2+1$ such that any subset $S \subset \overline{K}$ of size $2\delta^4k^2s^2p$ contains c satisfying $\Psi_{D,p,c}(\mathrm{sim}(C)) = \mathrm{sim}(\Psi_{D,p,c}(C))$.*

5.3 A Hitting Set (Proving Theorems 2 and 3)

Armed with Lemmas 17 and 18 we could now complete the construction of the hitting set for $\Sigma\Pi\Sigma\Pi_\delta(k,s,n)$ circuits using the faithful homomorphism Ψ with the right parameters.

Let $n,\delta,k,s \geq 1$ and let $r = R_\delta(k,s)$. We introduce the following parameters. They are blown up so that they support 2^k applications (one for each $I \subseteq [k]$) of Lemmas 14 and 18. Define $D = (D_1,D_2)$ by $D_1 := (2\delta n)^{2r}$ and $D_2 := \delta+1$, and $p_{\max} := 2^{2(k+1)} \cdot (2krsn\delta^2)^{8\delta^2+4\delta r}\lceil \log_2 D_1 \rceil^2 + 1$. Pick arbitrary $H_1, H_2 \subset \overline{K}$ of sizes $2^{k+2}k^2rs^2\delta^4 p_{\max}$ resp. $\delta s+1$. Finally, denote $\Psi^{(i)}_{D,p,c} := \Psi_{D,p,c}(x_i) \in \overline{K}[\boldsymbol{z}]$ for $i = 1,\ldots,n$ and define the subset

$$\mathcal{H}_{\delta,k,s} = \left\{ \left(\Psi^{(1)}_{D,p,c}(\boldsymbol{a}),\ldots,\Psi^{(n)}_{D,p,c}(\boldsymbol{a})\right) \;\middle|\; p \in [p_{\max}],\, c \in H_1,\, \boldsymbol{a} \in H_2^{r+1} \right\} \subset \overline{K}^n \quad .$$

The following theorem shows that, over large or zero characteristic, this is a hitting set for $\Sigma\Pi\Sigma\Pi_\delta(k,s,n)$ circuits.

Theorem 19. *Assume that $\mathrm{ch}(K) = 0$ or $\mathrm{ch}(K) > \delta^r$. Then $\mathcal{H}_{\delta,k,s}$ is a hitting set for $\Sigma\Pi\Sigma\Pi_\delta(k,s,n)$ circuits. It can be constructed in $\mathrm{poly}(\delta rsn)^{\delta^2 kr}$ time.*

Since trivially $R_\delta(2,s) = 1$, we obtain an explicit hitting set for the top fanin 2 case. Moreover, in this case we can also eliminate the dependence on the characteristic (because Lemma 18 is field independent).

Corollary 20. *Let K be of arbitrary characteristic. Then $\mathcal{H}_{\delta,2,s}$ is a hitting set for $\Sigma\Pi\Sigma\Pi_\delta(2,s,n)$ circuits. It can be constructed in $\mathrm{poly}(\delta sn)^{\delta^2}$ time.*

6 Conclusion

The notion of rank has been quite useful in depth-3 PIT. In this work we give the first generalization of it to depth-4 circuits. We used trdeg and developed fundamental maps – the faithful homomorphisms – that preserve trdeg of sparse polynomials in a blackbox and efficient way (assuming a small trdeg). Crucially, we showed that faithful homomorphisms preserve the nonzeroness of circuits.

Our work raises several open questions. The faithful homomorphism construction over a small characteristic has restricted efficiency, in particular, it is interesting only when the sparse polynomials have very low degree. Could Lemma 13 be improved to handle larger δ? In general, the classical methods stop short of dealing with small characteristic because the "geometric" Jacobian criterion is not there. We have given some new tools to tackle that, for eg., Corollary 5 and Lemmas 12 and 13. But more tools are needed, for eg. a homomorphism like that of Lemma 14 for arbitrary fields.

Currently, we do not know a better upper bound for $R_\delta(k, s)$ other than ks. For $\delta = 1$, it is just the rank of depth-3 identities, which is known to be $O(k^2 \log s)$ ($O(k^2)$ over $\mathbb{R}$) [25]. Even for $\delta = 2$ we leave the rank question open. We conjecture $R_2(k, s) = O_k(\log s)$ (generally, Conjecture 16). Our hope is that understanding these small δ identities should give us more potent tools to attack depth-4 PIT in generality.

References

1. Adleman, L.M., Lenstra, H.W.: Finding irreducible polynomials over finite fields. In: Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC), pp. 350–355 (1986)
2. Agrawal, M.: Proving lower bounds via pseudo-random generators. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 92–105. Springer, Heidelberg (2005)
3. Agrawal, M.: Determinant versus permanent. In: Proceedings of the 25th International Congress of Mathematicians (ICM), vol. 3, pp. 985–997 (2006)
4. Agrawal, M., Biswas, S.: Primality and identity testing via Chinese remaindering. Journal of the ACM 50(4), 429–443 (2003)
5. Agrawal, M., Vinay, V.: Arithmetic circuits: A chasm at depth four. In: Proceedings of the 49th Annual Symposium on Foundations of Computer Science (FOCS), pp. 67–75 (2008)
6. Anderson, M., van Melkebeek, D., Volkovich, I.: Derandomizing polynomial identity testing for multilinear constant-read formulae. In: Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC (2011)
7. Beecken, M., Mittmann, J., Saxena, N.: Algebraic Independence and Blackbox Identity Testing. Tech. Rep. TR11-022, Electronic Colloquium on Computational Complexity, ECCC (2011)
8. Chen, Z., Kao, M.: Reducing randomness via irrational numbers. SIAM J. on Computing 29(4), 1247–1256 (2000)
9. DeMillo, R.A., Lipton, R.J.: A probabilistic remark on algebraic program testing. Information Processing Letters 7(4), 193–195 (1978)
10. Dvir, Z., Gabizon, A., Wigderson, A.: Extractors and rank extractors for polynomial sources. Computational Complexity 18(1), 1–58 (2009)

11. Dvir, Z., Gutfreund, D., Rothblum, G., Vadhan, S.: On approximating the entropy of polynomial mappings. In: Proceedings of the 2nd Symposium on Innovations in Computer Science, ICS (2011)
12. Eisenbud, D.: Commutative Algebra with a View Toward Algebraic Geometry. Springer, New York (1995)
13. Heintz, J., Schnorr, C.P.: Testing polynomials which are easy to compute (extended abstract). In: Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing, New York, NY, USA, pp. 262–272 (1980)
14. Kabanets, V., Impagliazzo, R.: Derandomizing polynomial identity tests means proving circuit lower bounds. Computational Complexity 13(1), 1–46 (2004)
15. Kalorkoti, K.: A lower bound for the formula size of rational functions. SIAM J. Comp. 14(3), 678–687 (1985)
16. Karnin, Z., Shpilka, A.: Deterministic black box polynomial identity testing of depth-3 arithmetic circuits with bounded top fan-in. In: Proceedings of the 23rd Annual Conference on Computational Complexity (CCC), pp. 280–291 (2008)
17. Kayal, N.: The Complexity of the Annihilating Polynomial. In: Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC), pp. 184–193 (2009)
18. Kemper, G.: A Course in Commutative Algebra. Springer, Berlin (2011)
19. Lewin, D., Vadhan, S.: Checking polynomial identities over any field: Towards a derandomization? In: Proceedings of the 30th Annual Symposium on the Theory of Computing (STOC), pp. 428–437 (1998)
20. Lovász, L.: On determinants, matchings and random algorithms. In: Fundamentals of Computation Theory (FCT), pp. 565–574 (1979)
21. Lovász, L.: Singular spaces of matrices and their applications in combinatorics. Bol. Soc. Braz. Mat. 20, 87–99 (1989)
22. Perron, O.: Algebra I (Die Grundlagen). Berlin (1927)
23. Płoski, A.: Algebraic Dependence of Polynomials After O. Perron and Some Applications. In: Cojocaru, S., Pfister, G., Ufnarovski, V. (eds.) Computational Commutative and Non-Commutative Algebraic Geometry, pp. 167–173. IOS Press, Amsterdam (2005)
24. Saraf, S., Volkovich, I.: Black-box identity testing of depth-4 multilinear circuits. In: Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC (2011)
25. Saxena, N., Seshadhri, C.: From Sylvester-Gallai configurations to rank bounds: Improved black-box identity test for depth-3 circuits. In: Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS), pp. 21–29 (2010)
26. Saxena, N., Seshadhri, C.: An Almost Optimal Rank Bound for Depth-3 Identities. SIAM J. Comp. 40(1), 200–224 (2011)
27. Saxena, N., Seshadhri, C.: Blackbox identity testing for bounded top fanin depth-3 circuits: the field doesn't matter. In: Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC (2011)
28. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. Journal of the ACM 27(4), 701–717 (1980)
29. Shpilka, A., Yehudayoff, A.: Arithmetic Circuits: A survey of recent results and open questions. Foundations and Trends in Theoretical Computer Science 5(3-4), 207–388 (2010)
30. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: Ng, K.W. (ed.) EUROSAM 1979 and ISSAC 1979. LNCS, vol. 72, pp. 216–226. Springer, Heidelberg (1979)

A Fragment of ML Decidable by Visibly Pushdown Automata

David Hopkins[1,⋆], Andrzej S. Murawski[2,⋆⋆], and C.-H. Luke Ong[1]

[1] Department of Computer Science, University of Oxford, UK
[2] Department of Computer Science, University of Leicester, UK

Abstract. The simply-typed, call-by-value language, RML, may be viewed as a canonical restriction of Standard ML to ground-type references, augmented by a "bad variable" construct in the sense of Reynolds. By a *short* type, we mean a type of order at most 2 and arity at most 1. We consider the *O-strict* fragment of (finitary) RML, $\mathsf{RML}_{\text{O-Str}}$, consisting of terms-in-context $x_1 : \theta_1, \cdots, x_n : \theta_n \vdash M : \theta$ such that θ is short, and every argument type of every θ_i is short. $\mathsf{RML}_{\text{O-Str}}$ is surprisingly expressive; it includes several instances of (in)equivalence in the literature that are challenging to prove using methods based on (state-based) logical relations. We show that it is decidable whether a given pair of $\mathsf{RML}_{\text{O-Str}}$ terms-in-context is observationally equivalent. Using the fully abstract game semantics of RML, our algorithm reduces the problem to the language equivalence of visibly pushdown automata. When restricted to terms in canonical form, the problem is EXPTIME-complete.

1 Introduction

The standard approaches to the verification of higher-order programs are *type-based program analysis* on the one hand, and *theorem-proving* and *dependent types* on the other. The former is sound, but often imprecise; the latter typically requires human intervention. The paper is concerned with a relatively recent, game-semantics based approach to the verification of higher-order procedural programs. We consider a call-by-value language, RML, which has both functional and (stateful) imperative features, mediated by Church's simple type theory. RML may be viewed as a canonical restriction of Standard ML to ground-type references, except that it includes a "bad variable" construct [17,2].

Observational equivalence is a compelling notion of program equivalence. Two terms M and N are observationally equivalent, written $M \cong N$, if they are mutually replaceable in every program without changing the computational outcome. Because of the quantification over all program contexts, the theory of observational equivalence is rich and hard to reason about, as illustrated by the following example.

Example 1. (i) $\mathsf{let}\, c = \mathsf{ref}\ \mathsf{in}\ \lambda f^{\mathsf{unit}\to\mathsf{unit}}.(c := 1; f(); !c) \cong \lambda f^{\mathsf{unit}\to\mathsf{unit}}.(f(); 1)$
(ii) $\mathsf{let}\, c = \mathsf{ref}\ \mathsf{in}\ \lambda f^{\mathsf{unit}\to\mathsf{unit}}.(c := 0; f(); c := 1; f(); !c) \cong \lambda f^{\mathsf{unit}\to\mathsf{unit}}.(f(); f(); 1)$

⋆ Supported by Microsoft Research through its PhD Scholarship Programme.
⋆⋆ Supported by an EPSRC Advanced Research Fellowship (EP/C539753/1).

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 149–161, 2011.

(iii) $\mathsf{let}\, a = \mathsf{ref}\, \mathsf{in}\, \mathsf{let}\, r = \mathsf{ref}\, \mathsf{in}\, \lambda f.(r := !r + 1; a := f(!r); r := !r - 1; !a) \not\cong \lambda f.f(1)$

The two equivalences above, due to Pitts and Stark [16] and Thamsborg [3] respectively, are notoriously tricky to verify using methods based on (state-based) logical relations. The inequivalence, a somewhat surprising instance due to Stark [19], requires a rather delicate separating context to exhibit.

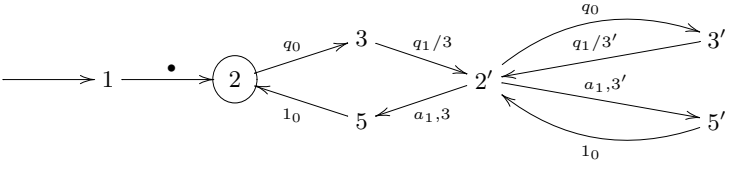
Let θ_i and θ range over RML types. We say that *observational equivalence is decidable at a type sequent* $\theta_1, \cdots, \theta_n \vdash \theta$ (or simply, $\overline{\theta} \vdash \theta$ is *decidable*) just if the following problem is decidable: given terms-in-context $x_1 : \theta_1, \cdots, x_n : \theta_n \vdash M, N : \theta$ of finitary RML (henceforth, written $\mathsf{RML_f}$), are they observationally equivalent? This paper is concerned with the question of classifying the decidable type sequents of $\mathsf{RML_f}$.

Following Ghica and McCusker [6], we use a method based on game semantics to decide observational equivalence of $\mathsf{RML_f}$. Take a term-in-context $\Gamma \vdash M : \theta$ with $\Gamma = x_1 : \theta_1, \cdots, x_n : \theta_n$. In game semantics [9,7], the term-in-context is interpreted as a P strategy $[\![\Gamma \vdash M : \theta]\!]$ for playing (against O, who takes the environment's perspective) in the prearena $[\![\overline{\theta} \vdash \theta]\!]$. A play between P and O is a sequence of moves in which each non-initial move has a justification pointer to some earlier move. Thanks to the fully abstract game semantics of RML, observational equivalence is characterised by *complete plays* i.e. $M \cong N$ iff the P-strategies, $[\![\Gamma \vdash M]\!]$ and $[\![\Gamma \vdash N]\!]$, trace out the same set of complete plays. (A play is complete if every question in it is subsequently answered in the play.) Strategies may be viewed as highly constrained processes, and are amenable to concrete, automata-theoretic representations. In certain prearenas—which we shall call *bi-strict*—plays may be represented simply by their underlying move sequence, because the justification pointers from both O- and P-moves are uniquely determined. Murawski studied bi-strict sequents in [11] and identified those that are decidable by reduction to the equivalence problem of deterministic FSA.

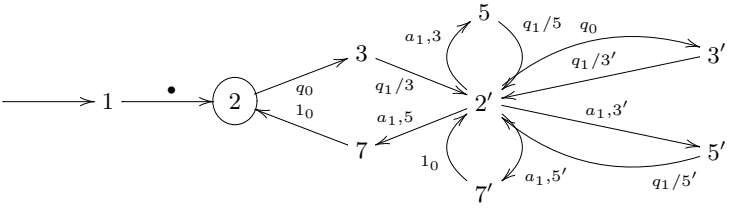
In this paper, we consider type sequents of $\mathsf{RML_f}$ that are *O-strict*. Plays over prearenas denoted by O-strict sequents enjoy the property that pointers from O-moves are uniquely determined by the underlying move sequence. We first give a simple characterisation of O-strict type sequents: $\theta_1, \cdots, \theta_n \vdash \theta$ is O-strict iff θ is short, and every argument type of every θ_i is short, where a type is said to be *short* if it has order at most 2 and arity at most 1. (Henceforth we write the O-strict fragment of $\mathsf{RML_f}$ as $\mathsf{RML_{O\text{-}Str}}$.) We then prove our first result: observational equivalence of $\mathsf{RML_{O\text{-}Str}}$ is decidable by reduction to the equivalence problem of visibly pushdown automata [4]. Our proof is by induction over the canonical forms of $\mathsf{RML_{O\text{-}Str}}$ terms-in-contexts. For each such term-in-context $\Gamma \vdash M$, we construct a visibly pushdown automaton $\mathcal{A}_{\Gamma \vdash M}$ that accepts the complete plays in the strategy denotation of the term, whereby P-questions are pushes, O-answers pops and all other moves no-ops. The key innovation of the construction lies in the encoding of the pointers from P-questions. Instead of trying to represent all such pointers present, we concentrate only on *representing a single pointer*. Note that for every pointer we need to represent, there must be an accepting run encoding its location. So even though each individual word accepted by the automaton may not have enough information to fully reconstruct the pointers, when we consider the full language we will be able to uniquely place all justification pointers. Our second result is that the observational equivalence of $\mathsf{RML_{O\text{-}Str}}$ terms-in-context in canonical form is EXPTIME-complete. EXPTIME-hardness is shown by a reduction of the equivalence

problem for nondeterministic automata on binary trees [18] to the problem of deciding observational equivalence at the sequent $\mathsf{unit} \to \mathsf{int} \vdash (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}$.

Example 2. Our algorithm can decide the (in)equivalence[1] instances in Example 1 as the terms belong to $\mathsf{RML}_{\text{O-Str}}$. (i) The VPA below represents the complete plays of the game semantics of the terms of Example 1(i). In the diagram, the edge label 'q/n' means 'on reading q, push n', and 'a, n' means 'on reading a and stack top is n, pop'.



(ii) This VPA represents the complete plays of the game semantic strategy for the terms of Example 1(ii).



Our results may be viewed as the first steps towards a complete classification of the decidable RML_f type sequents. In the case of (finitary) Idealized Algol, decidability (and if so, the complexity) of a given type sequent depends only on its type-theoretic order [12]. In contrast, the decidability of RML_f-sequents is not so neatly characterised by order (see the table below): there are undecidable sequents of order as low as 2 [11], amidst interesting classes of decidable sequents at each of orders 1 to 4. For comparison, we also give DFA-decidable (regular) and undecidable sequents [10,11].

RML_f-Fragment	Examples of Type Sequents (writing o for unit)
bi-strict, regular	$(o \to o) \to o \vdash o \to o$
bi-strict, not-reg.	$\vdash (o \to o) \to o$
$\mathsf{RML}_{\text{O-Str}}$	$((o \to \ldots \to o) \to o) \to o \vdash (o \to \ldots \to o) \to o$
undecidable	$\vdash (o \to o) \to (o \to o) \to o, \quad (((o \to o) \to o) \to o) \to o \vdash o$

The remaining open cases are those in which pointers from O-moves need to be represented explicitly. At the moment we see no way of dealing with them, as they seem to require potentially unbounded references to past computations. What we can say is that our method of single-pointer representation cannot be extended beyond the O-strict fragment of RML_f (as there are distinct strategies that have the same single-pointer representation).

[1] Restricted to sequents in which $\mathsf{int\ ref}$ does *not* occur, observational equivalence of RML conservatively extends that of Reduced ML (because mkvar is only needed at $\mathsf{int\ ref}$ for definability).

2 RML, Game Semantics and Visibly Pushdown Automata

RML is a call-by-value functional language with state [1]. It is similar to Reduced ML [16], the canonical restriction of Standard ML to ground-type references, except that it includes a "bad-variable" constructor (in the absence of the "bad-variable" constructor the equality test is definable). Types are generated by the grammar $\theta ::= \mathsf{unit} \mid \mathsf{int} \mid \mathsf{int\,ref} \mid \theta \to \theta$. The type assignment rules are completely standard. The operational semantics, which is defined as a "big-step" relation [11], is also standard. For closed terms, we write $M\Downarrow$ just if M reduces to some value. This can be used to define a natural notion of equivalence; intuitively, two terms are observationally equivalent if one can always replace the other without affecting the result of the computation. Given two terms-in-context $\Gamma \vdash M_1, M_2 : \theta$, we say that M_1 *observationally approximates* M_2 (written $\Gamma \vdash M_1 \sqsubseteq M_2$) if for all contexts $C[-]$ such that $C[M_1]$ and $C[M_2]$ are closed terms of type unit, we have that if $C[M_1]\Downarrow$ then $C[M_2]\Downarrow$. We say M_1 and M_2 are *observationally equivalent* (written $\Gamma \vdash M_1 \cong M_2$) if $\Gamma \vdash M_1 \sqsubseteq M_2$ and $\Gamma \vdash M_2 \sqsubseteq M_1$. It can be shown that every RML term is effectively convertible to an equivalent term in *canonical form*, defined by the following grammar ($\beta \in \{\mathsf{unit}, \mathsf{int}\}$).

$$\begin{aligned}\mathbb{C} ::= {} & () \mid i \mid x^{\beta} \mid x^{\beta}\ op\ y^{\beta} \mid \mathsf{if}\ x^{\beta}\ \mathsf{then}\ \mathbb{C}\ \mathsf{else}\ \mathbb{C} \mid x^{\mathsf{int\,ref}} := y^{\mathsf{int}} \mid !x^{\mathsf{int\,ref}} \mid \lambda x^{\theta}.\mathbb{C} \mid \\ & \mathsf{mkvar}(\lambda x^{\mathsf{unit}}.\mathbb{C}, \lambda y^{\mathsf{int}}.\mathbb{C}) \mid \mathsf{let}\ x = \mathsf{ref}\ \mathsf{in}\ \mathbb{C} \mid \mathsf{while}\ \mathbb{C}\ \mathsf{do}\ \mathbb{C} \mid \mathsf{let}\ x^{\beta} = \mathbb{C}\ \mathsf{in}\ \mathbb{C} \mid \\ & \mathsf{let}\ x = zy^{\beta}\ \mathsf{in}\ \mathbb{C} \mid \mathsf{let}\ x = z\,\mathsf{mkvar}(\lambda u^{\mathsf{unit}}.\mathbb{C}, \lambda v^{\mathsf{int}}.\mathbb{C})\ \mathsf{in}\ \mathbb{C} \mid \mathsf{let}\ x = z(\lambda x^{\theta}.\mathbb{C})\ \mathsf{in}\ \mathbb{C}\end{aligned}$$

In order to achieve a decidability result, we consider the *finitary* fragment of RML, written $\mathsf{RML_f}$. That is, we restrict the type int to be a finite subset of $\mathbb{Z}$.

Call-By-Value Game Semantics We present call-by-value game semantics in the style of Honda and Yoshida [7], as opposed to Abramsky and McCusker's isomorphic model [1], as Honda and Yoshida's constructions are more concrete, lead to more compact alphabets, and so are more suited to algorithmic analysis.

An *arena* A is a triple $(M_A, \vdash_A, \lambda_A)$ where M_A is a set of *moves* where $I_A \subseteq M_A$ consists of *initial* moves, $\vdash_A \subseteq M_A \times (M_A \backslash I_A)$ is called the *justification relation*, and $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$ a labelling function such that for all $i_A \in I_A$ we have $\lambda_A(i_A) = (P, A)$ and if $m \vdash_A m'$ then $(\pi_1\lambda_A)(m) \neq (\pi_1\lambda_A)(m')$ and $(\pi_2\lambda_A)(m') = A \Rightarrow (\pi_2\lambda_A)(m) = Q$.

The function λ_A labels moves as belonging to either *Opponent* or *Proponent* and as being either a *Question* or an *Answer*. Note that answers are always justified by questions, but questions can be justified by either a question or an answer. We will use arenas to model types. However, the actual games will be played over *prearenas*, which are defined in the same way except that initial moves are O-questions.

Three basic arenas are 0, the empty arena, 1, the arena containing a single initial move $\bullet$, and $\mathbb{Z}$, which has the integers as its set of moves, all of which are initial P-answers.

Some constructions on arenas are described below. Here we use $\overline{I_A}$ as an abbreviation for $M_A \backslash I_A$, and $\overline{\lambda_A}$ for the O/P-complement of λ_A. Intuitively $A \otimes B$ is the union of the arenas A and B, but with the initial moves combined pairwise. $A \Rightarrow B$ is slightly more complex. First we add a new initial move, $\bullet$. We take the O/P-complement of A, change the initial moves into questions, and set them to now be justified by $\bullet$. Finally,

we take B and set its initial moves to be justified by A's initial moves. The final construction, $A \to B$, takes two arenas A and B and produces a prearena, as shown below. This is essentially the same as $A \Rightarrow B$ without the initial move $\bullet$.

$$M_{A \Rightarrow B} = \{\bullet\} \uplus M_A \uplus M_B \qquad M_{A \otimes B} = I_A \times I_B \uplus \overline{I_A} \uplus \overline{I_B}$$

$$I_{A \Rightarrow B} = \{\bullet\} \qquad I_{A \otimes B} = I_A \times I_B$$

$$\lambda_{A \Rightarrow B} = m \mapsto \begin{cases} PA & \text{if } m = \bullet \\ OQ & \text{if } m \in I_A \\ \overline{\lambda_A}(m) & \text{if } m \in \overline{I_A} \\ \lambda_B(m) & \text{if } m \in M_B \end{cases} \qquad \lambda_{A \otimes B} = m \mapsto \begin{cases} PA & \text{if } m \in I_A \times I_B \\ \lambda_A(m) & \text{if } m \in \overline{I_A} \\ \lambda_B(m) & \text{if } m \in \overline{I_B} \end{cases}$$

$$\vdash_{A \Rightarrow B} = \{(\bullet, i_A) | i_A \in I_A\} \cup \{(i_A, i_B) | i_A \in I_A, i_B \in I_B\} \cup \vdash_A \cup \vdash_B$$

$$\vdash_{A \otimes B} = \{((i_A, i_B), m) | i_A \in I_A \wedge i_B \in I_B \wedge (i_A \vdash_A m \vee i_B \vdash_B m)\} \cup (\vdash_A \cap (\overline{I_A} \times \overline{I_A})) \cup (\vdash_B \cap (\overline{I_B} \times \overline{I_B}))$$

$$M_{A \to B} = M_A \uplus M_B \qquad \lambda_{A \to B}(m) = \begin{cases} OQ & \text{if } m \in I_A \\ \overline{\lambda_A}(m) & \text{if } m \in \overline{I_A} \\ \lambda_B(m) & \text{if } m \in M_B \end{cases}$$

$$I_{A \to B} = I_A \qquad \vdash_{A \to B} = \{(i_A, i_B) | i_A \in I_A, i_B \in I_B\} \cup \vdash_A \cup \vdash_B$$

We intend arenas to represent types, in particular $[\![\mathsf{unit}]\!] = 1$, $[\![\mathsf{int}]\!] = \mathbb{Z}$ (or a finite subset of $\mathbb{Z}$ for $\mathsf{RML_f}$) and $[\![\theta_1 \to \theta_2]\!] = [\![\theta_1]\!] \Rightarrow [\![\theta_2]\!]$. A term $x_1 : \theta_1, \ldots, x_n : \theta_n \vdash M : \theta$ will be represented by a *strategy* for the prearena $[\![\theta_1]\!] \otimes \ldots \otimes [\![\theta_n]\!] \to [\![\theta]\!]$.

A *justified sequence* in a prearena A is a sequence of moves from A in which the first move is initial and all other moves m are equipped with a pointer to an earlier move m', such that $m' \vdash_A m$.

A *play* s is a justified sequence which additionally satisfies the following conditions.

(i) *Alternation*: O and P take it in turns to play moves. That is if $t\,m\,m' \sqsubseteq s$ then $\lambda^{OP}(m) \neq \lambda^{OP}(m')$.

(ii) *Well-Bracketing*: Questions asked first must be answered first. If $t\,q\,\widehat{t'}\,a \sqsubseteq s$ then all questions in t' must be answered in t'.

(iii) *Visibility*: If $t\,m\,\widehat{t'}\,m' \sqsubseteq s$ then m appears in $\mathsf{view}(t\,m\,t')$, where view is defined by, $\mathsf{view}(\epsilon) = \epsilon$, $\mathsf{view}(o) = o$ if o is initial, and $\mathsf{view}(t\,m\,\widehat{t'}\,m') = \mathsf{view}(t)\,m\,m'$.

We denote the set of all valid plays over prearena A as P_A.

A *strategy* σ for prearena A is a non-empty, even-prefix-closed set of plays from A, satisfying the condition that if $s\,m_1, s\,m_2 \in \sigma$ then $s\,m_1 = s\,m_2$.

We can think of a strategy as being a playbook telling P how to respond by mapping odd-length plays to moves.

A play is *complete* if all questions have been answered. Note that (unlike in the call-by-name case) a complete play is not necessarily maximal. We denote the set of complete plays in strategy σ by $\mathsf{comp}(\sigma)$.

Game Semantics of RML In the game semantic model of RML, a term-in-context $x_1 : \theta_1, \ldots, x_n : \theta_n \vdash M : \theta$ is represented by a strategy for the prearena $[\![\theta_1]\!] \otimes \ldots \otimes [\![\theta_n]\!] \to [\![\theta]\!]$. These strategies are built up compositionally over the syntax of the term. Essentially, free identifiers $x : \theta \vdash x : \theta$ are interpreted as *copy-cat* strategies where P always copies O's move into the other copy of $[\![\theta]\!]$, $\lambda x.M$ allows multiple copies of

$[\![M]\!]$ to be run, application MN requires a form of parallel composition plus hiding and the other constructions can be interpreted using special strategies. The game semantic model is fully abstract in the following sense.

Theorem 1 (Abramsky and McCusker 1997 [1,2]). *For all* RML*-terms-in-context* $\Gamma \vdash M, N : \theta$*, we have* $M \sqsubseteq N$ *iff* $\mathsf{comp}([\![\Gamma \vdash M]\!]) \subseteq \mathsf{comp}([\![\Gamma \vdash N]\!])$.

We will show decidability of observational equivalence for a fragment of RML by representing the game semantics of terms as *Visibly Pushdown Automata* (VPA) [4]. VPA are a subclass of pushdown automata in which the stack action is uniquely determined by the input letter. The alphabet is partitioned into push-letters, pop-letters and noop-letters. On reading a letter the automaton must perform the appropriate action. (We write $s \xrightarrow{q/x} s'$ to mean "on reading q, push x" and $s \xrightarrow{a,x} s'$ to mean "on reading a and stack top is x, pop".) This gives them very attractive closure properties. In particular, equivalence of deterministic VPA is decidable in polynomial time.

3 Characterising the O-Strict Fragment of RML

In order to represent strategies using automata, we need to be able to encode plays (move sequence with pointers) as words. In some cases pointers can be uniquely reconstructed from the underlying move sequence, thanks to the visibility or well-bracketing conditions, which constrain the position of the justifying move. For instance, the targets of pointers from answer-moves can always be deduced from the underlying sequence of moves. In general, however, pointers must be encoded explicitly, and this poses a representational challenge because the target of a pointer can be arbitrarily far back in the history of the play. We say that a play is *O-strict* just if the pointer from every O-question in the play is uniquely determined by the underlying move sequence. A prearena is said to be *O-strict* if every play of the prearena is O-strict; a type sequent is *O-strict* if its denotation (in the call-by-value game semantics) is an O-strict prearena. It follows that when representing plays of an O-strict prearena, only pointers from P-questions need to be encoded. In this section, we aim to find a simple characterisation of the O-strict sequents.

Every RML-type θ can be written uniquely as $\theta_1 \to \cdots \to \theta_n \to \beta$ (by convention $\to$ associates to the right), where $n \geq 0$ and β stands for unit, int or $\mathsf{int\ ref}$. In what follows we shall write $(\theta_1, \cdots, \theta_n, \beta)$ for θ. The *arity*, $ar(\theta)$, and *order*, $ord(\theta)$, of θ are defined as follows. $ar(\theta) := \begin{cases} n & \text{if } \beta = \mathsf{unit} \text{ or } \mathsf{int} \\ n+1 & \text{if } \beta = \mathsf{int\ ref}. \end{cases}$ $ord(\mathsf{unit}) = ord(\mathsf{int}) = 0$, $ord(\mathsf{int\ ref}) = 1$ and $ord(A \to B) = \max(ord(A) + 1, ord(B))$. For clarity, we shall assume $\beta = \mathsf{unit}$ in the argument that follows; there is no loss of generality because essentially identical considerations work for the case of int, and $\mathsf{int\ ref}$ can be treated as $\mathsf{unit} \to \mathsf{unit}$.

Types on the right of O-strict sequents. Consider an arena with the following enabling chain $q_0 \vdash a_0 \vdash q_1 \vdash a_1 \vdash q_2$. (For brevity, we shall say that the arena has a $qaqaq$-branch.) Then sequences of the form $q_0 a_0 (q_1 a_1)^n q_2$, where $n \geq 0$, are all plays, regardless of which occurrence of a_1 is used to justify q_2. Representing the pointer from the O-question q_2 would seem to require unbounded memory or an infinite alphabet.

Observe that the prearena of the type sequent $\vdash (\mathsf{unit}, \mathsf{unit}, \mathsf{unit})$ has a $qaqaq$-branch. In general, the same is the case for $\Gamma \vdash (\theta_1, \cdots, \theta_k, \mathsf{unit})$, where $k \geq 2$. In other words, the type on the right of an O-strict sequent has the shape (θ, unit) or is unit. Another troublesome sequent is $\vdash (((\mathsf{unit}, \mathsf{unit}), \mathsf{unit}), \mathsf{unit})$ which has a $qaqqq$-branch. In general, types of the form $((\theta_1, \cdots, \theta_k, \mathsf{unit}), \mathsf{unit})$ have a similar problem in case θ_i is functional for some $1 \leq i \leq k$. Thus, types on the right of an O-strict sequent must be of type Θ_2 (we shall call a type *short* just if it is in Θ_2) where

$$\Theta_1 \ ::= \ \mathsf{unit} \mid \mathsf{unit} \to \Theta_1 \qquad\qquad \Theta_2 \ ::= \ \mathsf{unit} \mid \Theta_1 \to \mathsf{unit}.$$

Equivalently, a type is in Θ_2 just if it has order at most 2 and arity at most 1.

Types on the left of O-strict sequents. Type sequents that contain $((\mathsf{unit}, \mathsf{unit}, \mathsf{unit}), \mathsf{unit})$ on the left are similarly problematic because the corresponding prearenas have a $qqqaq$-branch. Generally, sequents of the shape $\cdots, (\theta_1, \cdots, \theta_k, \mathsf{unit}), \cdots \vdash \cdots$ are not O-strict, if for some i, $\theta_i = (\theta_i^1, \cdots, \theta_i^{k'}, \mathsf{unit})$ and $k' \geq 2$.

Sequents that have the type $(((\mathsf{unit}, \mathsf{unit}), \mathsf{unit}), \mathsf{unit}), \mathsf{unit})$ on the left are also not O-strict because the corresponding prearenas have a $qqqqq$ branch. In general, this is the case for $\theta_i^1 = (\alpha_i^1, \cdots, \alpha_i^{k''}, \mathsf{unit})$, whenever some α_i^j is functional. Hence, if a sequent is O-strict, then each θ_i^1 must be of type Θ_1, i.e. each θ_i must be in Θ_2. This leads us to the class $\Theta_3 \ ::= \ \mathsf{unit} \mid \Theta_2 \to \Theta_3$. Equivalently a type is in Θ_3 just if it has shape $(\theta_1, \cdots, \theta_k, \mathsf{unit})$ where $k \geq 0$ and $\theta_i \in \Theta_2$ for each i. Note that Θ_3 contains Θ_1 but not Θ_2.

Lemma 1. *A type sequent, $\theta_1, \cdots, \theta_n \vdash \theta$, is O-strict iff $\theta \in \Theta_2$, and each $\theta_i \in \Theta_3$.*

So far we have omitted int and $\mathsf{int\ ref}$. To incorporate them into the characterisation, we treat int in the same way as unit, and $\mathsf{int\ ref}$ in the same way as $\mathsf{unit} \to \mathsf{unit}$. The revised definition of the collections, Θ_2 and Θ_3, thus reads as follows.

$$\begin{array}{lll} \Theta_0 \ ::= \ \mathsf{unit} \mid \mathsf{int} & & \Theta_2 \ ::= \ \Theta_0 \mid \Theta_1 \to \Theta_0 \mid \mathsf{int\ ref} \\ \Theta_1 \ ::= \ \Theta_0 \mid \Theta_0 \to \Theta_1 \mid \mathsf{int\ ref} & & \Theta_3 \ ::= \ \Theta_0 \mid \Theta_2 \to \Theta_3 \mid \mathsf{int\ ref} \end{array}$$

Definition 1. The *O-strict fragment of* RML, henceforth referred to as $\mathsf{RML}_{\text{O-Str}}$, consists of terms-in-context of the shape $x_1 : \Theta_3, \cdots, x_n : \Theta_3 \vdash M : \Theta_2$.

Since conversion to canonical form preserves types, canonical forms of $\mathsf{RML}_{\text{O-Str}}$-terms also belong to $\mathsf{RML}_{\text{O-Str}}$. Consequently, they satisfy the following properties.

(i) If $\Gamma \vdash \lambda x.\mathbb{C}$ is in $\mathsf{RML}_{\text{O-Str}}$, then $\Gamma, x : \Theta_1 \vdash \mathbb{C} : \Theta_0$.
(ii) If $\Gamma \vdash \mathsf{let}\, x = \mathsf{ref}\, \mathsf{in}\, \mathbb{C}$ is in $\mathsf{RML}_{\text{O-Str}}$, then $\Gamma, x \vdash \mathbb{C} : \Theta_2$.
(iii) If $\Gamma \vdash \mathsf{let}\, x = \cdots \mathsf{in}\, \mathbb{C}$ is in $\mathsf{RML}_{\text{O-Str}}$, then $\Gamma, x : \Theta_3 \vdash \mathbb{C} : \Theta_2$.
(iv) If $\Gamma \vdash \mathsf{let}\, x = z(\lambda y.\mathbb{C})\, \mathsf{in}\, \cdots$ is in $\mathsf{RML}_{\text{O-Str}}$, then $\Gamma, y : \Theta_1 \vdash \mathbb{C} : \Theta_0$.

Example 3. The following are $\mathsf{RML}_{\text{O-Str}}$ terms-in-contexts.

(i) $\begin{cases} f : \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit} \vdash \mathsf{let}\, g = f()\, \mathsf{in}\, (\mathsf{let}\, h = f()\, \mathsf{in}\, g()) : \mathsf{unit} \\ f : \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit} \vdash \mathsf{let}\, g = f()\, \mathsf{in}\, (\mathsf{let}\, h = f()\, \mathsf{in}\, h()) : \mathsf{unit} \end{cases}$
(ii) $\begin{cases} f : ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit} \vdash f(\lambda x^{\mathsf{unit}\to\mathsf{unit}}.f(\lambda y^{\mathsf{unit}\to\mathsf{unit}}.x())) : \mathsf{unit} \\ f : ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit} \vdash f(\lambda x^{\mathsf{unit}\to\mathsf{unit}}.f(\lambda y^{\mathsf{unit}\to\mathsf{unit}}.y())) : \mathsf{unit} \end{cases}$
(iii) The three pairs of terms in Example 1.

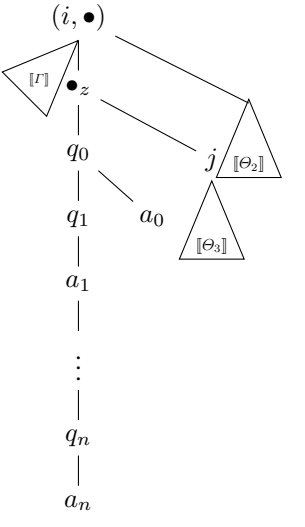
4 The O-Strict Fragment is VPA-Decidable

We shall show that the (fully abstract) game semantics of every $\mathsf{RML}_{\text{O-Str}}$-term can be faithfully represented using VPAs in the following sense.

Theorem 2. *There is an algorithm that transforms a given* $\mathsf{RML}_{\text{O-Str}}$*-term-in-context* $\Gamma \vdash M : \theta$ *to a VPA* $\mathcal{A}_{\Gamma\vdash M}$ *such that* $\Gamma \vdash M_1 \cong M_2$ *iff* $\mathcal{L}(\mathcal{A}_{\Gamma\vdash M_1}) = \mathcal{L}(\mathcal{A}_{\Gamma\vdash M_2})$.

Proof Outline. We wish to show that for all $\mathsf{RML}_{\text{O-Str}}$-terms $\Gamma \vdash M : \theta$ there exists (constructively) a VPA $\mathcal{A}_{\Gamma\vdash M}$ that accepts (some representation of) $[\![\Gamma \vdash M : \theta]\!]$.

To simplify this, we define $[\![\ldots]\!]_i$ by $[\![\Gamma \vdash M : \theta]\!] := \sum_{i\in I_{[\![\Gamma]\!]}} i\,[\![\Gamma \vdash M : \theta]\!]_i$. (To save space, we write $[\![\Gamma \vdash M : \theta]\!]$ simply as $[\![M]\!]$.) That is, $[\![M]\!]_i$ contains all plays of $[\![M]\!]$ which begin with initial move i, but with i removed. We will define automata $\mathcal{A}^i_M$ which accept the underlying move sequences of all complete plays in $[\![M]\!]_i$. To complete the proof, we will need to encode justification pointers, but for now we omit them. We partition our alphabet so that all P-questions are pushes, all O-answers pops and everything else noops.



Our construction proceeds inductively over the canonical forms. The simpler canonical forms can be described using regular expressions or as straightforward combinations of their subautomata. The case of λ-abstraction requires using the stack to nest copies of the body of the function. The construction for $\mathsf{let}\, x = \mathsf{ref}\, \mathsf{in}\, M$ stores the value of the variable in the state. The most complicated cases are those of the form $\mathsf{let}\, x = zM\, \mathsf{in}\, N$ and here we consider the hardest of them, $\mathsf{let}\, x = z(\lambda y.M)\, \mathsf{in}\, N$. The relevant prearena is shown in the figure on the right.

We assume the automata $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$ and $\mathcal{A}^{(i,j)}_{\Gamma,x\vdash N}$. To construct $\mathcal{A}^i_{\Gamma\vdash \mathsf{let}\, x=z(\lambda y.M)\, \mathsf{in}\, N}$ we take as our set of states:

$$Q^i_{\Gamma\vdash \mathsf{let}\, x=z(\lambda y.M)\, \mathsf{in}\, N} = \{(1),(2)\} \uplus \biguplus_{q_0\in I_{[\![\theta_1]\!]}} Q^{(i,q_0)}_{\Gamma,y\vdash M} \uplus \biguplus_{j\in I_{[\![\theta_3]\!]}} Q^{(i,j)}_{\Gamma,x\vdash N} \uplus \biguplus_{q_0\in I_{[\![\theta_1]\!]}, j\in I_{[\![\theta_3]\!]}} \left(Q^{(i,q_0)}_{\Gamma,y\vdash M} \times \widehat{Q^{(i,j)}_{\Gamma,x\vdash N}}\right)$$

where $\widehat{Q^{(i,j)}_{\Gamma,x\vdash N}}$ is the set of states s in $\mathcal{A}^{(i,j)}_{\Gamma,x\vdash N}$ such that $t \xrightarrow{m_x} s$ is a transition, with m_x a P-move in $[\![\Theta_3]\!]$. (1) is the initial state and the final states are those from $\mathcal{A}^{(i,j)}_{\Gamma,x\vdash N}$. The set of stack symbols is the disjoint union of the stack symbols used in the automata for M and N, plus the fresh symbol (1), plus the states of each $Q^{(i,q_0)}_{\Gamma,y\vdash M}$.

Large sections of the play will proceed as in $[\![M]\!]$ or $[\![N]\!]$. In particular, when in a $Q^{(i,j)}_{\Gamma,x\vdash N}$-state, play proceeds as in $\mathcal{A}^{(i,j)}_{\Gamma,x\vdash N}$ (although we will add additional transitions).

Similarly, when in a state with a $Q^{(i,q_0)}_{\Gamma,y\vdash M}$ component, play continues as in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$ (although non-$[\![\Gamma]\!]$-transitions will be redirected). Hence we have that if $s_M \overset{m\Box}{\to} t_M$ in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$ where m is a $[\![\Gamma]\!]$-move then in our new automaton we have $s_M \overset{m\Box}{\to} t_M$ and $(s_M, s_N) \overset{m\Box}{\to} (t_M, s_N)$ for all $s_N \in \widehat{Q^{(i,j)}_{\Gamma,x\vdash N}}$. Similarly, if $s_N \overset{m\Box}{\to} t_N$ in $\mathcal{A}^{(i,j)}_{\Gamma,x\vdash N}$ then we have $s_N \overset{m\Box}{\to} t_N$. Here we use $\overset{m\Box}{\to}$ to represent that this could be a push-, pop- or a noop-transition but whatever the case the transitions in the new automaton will perform the same stack action as in the old one.

The initial section of the play will correspond to evaluating $z(\lambda y.M)$. After the initial move, P will play $\bullet_z$. At this point, O can either play an initial $[\![\Theta_3]\!]$ move j, or play q_0, opening an M-thread. If O chooses the latter, play proceeds as in $[\![M]\!]$ until P plays in $[\![\Theta_1 \to \Theta_0]\!]$ (that is either P plays a_0, closing the M-thread, or some q_i). At this point O can choose to continue the current M-thread (unless P has closed it by playing a_0) or to open a new M-thread with q_0. Note that if O opens a new M-thread, while the old one is still open, the old thread will be left in a position where the only valid move is for O to answer the pending q_i with a_i. Thus bracketing ensures that we cannot revisit an old M-thread until we have closed the current one.

The transitions needed to represent this section of the play are:

- $(1) \overset{\bullet_z/(1)}{\to} (2)$.
- $(2) \overset{q_0}{\to} i^{q_0}_M$ where $i^{q_0}_M$ is the initial state in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$.
- If $s_M \overset{a_0}{\to} t_M$ in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$, then $s_M \overset{a_0}{\to} (2)$.
- If $s_M \overset{q_i/\gamma}{\to} t_M \overset{a_i,\gamma}{\to} u_M$, $i > 0$, in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$ (note that this must be the only transition out of s_M), then $s_M \overset{q_i/(s_M)}{\to} (2)$ and $(2) \overset{a_i,(s_M)}{\to} u_M$.

Eventually we may reach a point where all M-threads are closed and O plays j, for which we have transitions $(2) \overset{j,(1)}{\to} i^j_N$ where i^j_N is the initial state in $\mathcal{A}^{(i,j)}_{\Gamma,x\vdash N}$. Play then proceeds as in $[\![N]\!]$, except that if P ever plays in x (that is in $[\![\Theta_3]\!]$), then O again gets the chance to play q_0 and open an M-thread. If this happens then as before the threads can be stacked. Further, whenever O has the chance to open a new M-thread, O also has the option of resuming play in N by playing in $[\![\Theta_3]\!]$. If there are currently open M-threads when O chooses to return to N, then to obey bracketing it must be a $[\![\Theta_3]\!]$-question which O plays. As before, the only way to resume an open M-thread is with an answer, so to obey bracketing this can only happen after the $[\![\Theta_3]\!]$-question is answered. To manage this, for all $s_N \in \widehat{Q^{(i,j)}_{\Gamma,x\vdash N}}$ we need to have the following transitions:

- $s_N \overset{q_0}{\to} (i^{q_0}_M, s_N)$, where $i^{q_0}_M$ is the initial state in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$.
- If $s_M \overset{a_0}{\to} t_M$ in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$, then $(s_M, s_N) \overset{a_0}{\to} s_N$.
- If $s_M \overset{q_i/\gamma}{\to} t_M \overset{a_i,\gamma}{\to} u_M$, $i > 0$, in $\mathcal{A}^{(i,q_0)}_{\Gamma,y\vdash M}$ then $(s_M, s_N) \overset{q_i/(s_M)}{\to} s_N$ and $s_N \overset{a_i,(s_M)}{\to} (u_M, s_N)$.

Remark 1. It follows from the construction that $\mathcal{A}_{\Gamma\vdash M:\theta}$ is regular if types of free variables are $((\mathsf{unit},\mathsf{unit}),\cdots,(\mathsf{unit},\mathsf{unit}),\mathsf{unit})$ and the type of M is $(\mathsf{unit},\mathsf{unit})$ or unit.

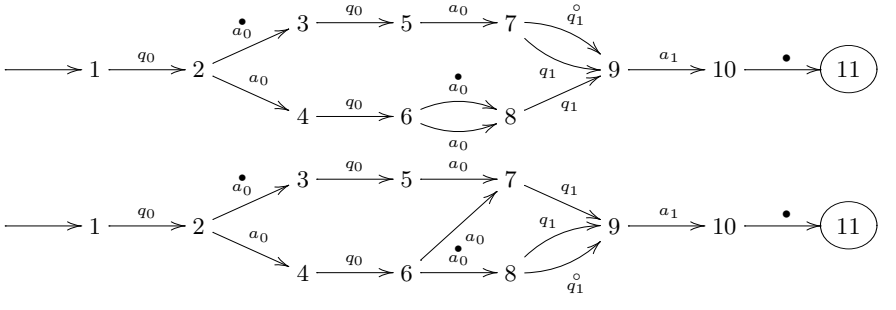
Pointers. We now consider how to represent pointers. Since we are concerned with O-strict prearenas, we only try to encode pointers from P-moves. Further, instead of describing the location of every pointer in a play, in each run of the automaton we only give the position of a single pointer. However, for every pointer we need to represent there must be an accepting run encoding its location. Since our strategies are deterministic, each P-move has a unique justifier and so when we consider the full language accepted by the automaton this encoding scheme gives us sufficient information to reconstruct all justification pointers.

If $s\,m\,s'ns''$ is a sequence of moves, we will use $s\ \overset{\bullet}{m}\ s'\ \overset{\circ}{n}\ s''$ to represent that there is a pointer from (the P-move) n to m. We refer to moves tagged with $\bullet$ as target-moves and those tagged with $\circ$ as source-moves. We will construct automata $\mathcal{A}^i_M$ which accept all strings that are either the underlying move sequence of a complete play in $[\![M]\!]_i$ or the underlying move sequence plus the encoding of a single justification pointer from a P question. Note that as we omit the initial move, we cannot encode pointers that point to it. However, this is not a problem since there is only ever one occurrence of the initial move so any pointers to it are always uniquely reconstructible. All other justification pointers from P-questions must have a representation in the automaton's language. Note that if $\mathcal{L}(\mathcal{A}^i_{\Gamma\vdash M}) = \mathcal{L}(\mathcal{A}^i_{\Gamma\vdash N})$ for all $i \in I_{[\![\Gamma]\!]}$ then $\mathsf{comp}([\![\Gamma \vdash M]\!]) = \mathsf{comp}([\![\Gamma \vdash N]\!])$.

In the case of $\mathsf{let}\, x = z(\lambda y.M)\,\mathsf{in}\, N$ we must ensure we preserve all pointers from $[\![M]\!]$ and $[\![N]\!]$, plus that in each $[\![M]\!]$-thread q_1 can point to the q_0 that opened that thread and finally that if in $[\![N]\!]$ P plays an $[\![x]\!]$-move justified by j, this can point at the copy of j which started $[\![N]\!]$. We must also take care to enforce that each accepting run only contains the encoding of a single pointer.

Example 4. (i) Take the term $\mathsf{let}\, g = f()\,\mathsf{in}\,(\mathsf{while}\, b()\,\mathsf{do}\,(\mathsf{let}\, h = f()\,\mathsf{in}\,()));g()$, where $f : \mathsf{unit} \to \mathsf{unit} \to \mathsf{unit}$ and $b : \mathsf{unit} \to \mathsf{int}$. The corresponding automaton will accept the following sequences: $q_f a_f (q_b 1_b q_f a_f)^* q_b 0_b q_{f'} a_{f'} a$ (no pointer information), $q_f\ \overset{\bullet}{a_f}\ (q_b 1_b q_f a_f)^* q_b 0_b q_{f'} a_{f'} a$ and $q_f a_f (q_b 1_b q_f a_f)^* q_b 1_b q_f\ \overset{\bullet}{a_f}\ (q_b 1_b q_f a_f)^* q_b 0_b q_{f'} a_{f'} a$ (information about single possible targets), and $q_f\ \overset{\bullet}{a_f}\ (q_b 1_b q_f a_f)^* q_b 0_b\ \overset{\circ}{q_{f'}}\ a_{f'} a$ (a single pointer is represented). Note that any occurrence of a_f could be a potential target for the pointer from $q_{f'}$. By annotating moves with $\bullet$ and $\circ$ we avoid the need for unbounded indices that would otherwise have to be used to represent pointers.

(ii) These automata represent the complete plays of the strategies for the terms of Example 3(i). As the language is regular we hide the stack actions. The underlying move sequences are identical but the encoding of pointers allows us to differentiate them.

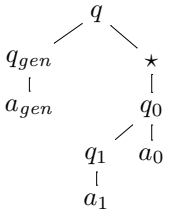


5 Complexity

Following [15], we define the size of a VPA to be the sum of the number of states and the number of stack symbols. The size of the alphabet is linear in the size of the input word and so we ignore it. The number of transitions is bounded by a polynomial in the size of the automaton.

In each case of the construction, the set of states consists of a number of fresh states, a number of copies of the states from sub-automata and a number of copies of pairs of states from different sub-automata. The set of stack symbols is similar. This means, that if automaton $\mathcal{A}_M$ is built up from n sub-automata $\mathcal{A}_{M_1} \ldots \mathcal{A}_{M_n}$ then $|\mathcal{A}_M| \leq c \times \left(1 + \sum |\mathcal{A}_{M_i}| + \sum_{i \neq j} |\mathcal{A}_{M_i}| \times |\mathcal{A}_{M_j}|\right)$, for some constant c. Given that at each step the size of the problem is greater than the sum of the size of the sub-problems, the implied recursion has an exponential bound.

Now the time required to construct each automaton is polynomial in its size (so exponential in the size of the input). The time taken to check whether two deterministic VPA are equivalent is polynomial in the size of the two VPA. Finally, the number of VPA we will need to check is exponential in the size of the input (in the number of int-components in the context). Altogether, this gives an exponential bound on the total amount of time required to check two $\mathsf{RML}_{\text{O-Str}}$-terms in canonical form for observational equivalence.



It turns out that this bound is optimal. One can show EXPTIME-hardness using a reduction of the EXPTIME-complete equivalence problem for nondeterministic automata on binary trees [18]. Through that route, we can show that observational equivalence is EXPTIME-hard for canonical terms $gen : \mathsf{unit} \to \mathsf{int} \vdash \mathbb{C} : (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}$. The associated arena A is shown on the right.

In order to represent (ranked) binary trees, let us assume that values of type int are partitioned into the set of binary and nullary labels, ranged over l_2 and l_0 respectively. Then any ranked binary tree T over such labels can be represented by the play $q \star \mathcal{S}(T)$ on A, where $\mathcal{S}(T)$ is defined as $\mathcal{S}(l) := q_0\, q_{gen}\, l_{gen}\, a_0$; $\mathcal{S}(n(T_1, T_2)) := q_0\, q_{gen}\, n_{gen}\, q_1\, \mathcal{S}(T_1)\, a_1\, q_1\, \mathcal{S}(T_2)\, a_1\, a_0$. Note that $\mathcal{S}(T)$ can be seen as a record of a depth-first traversal of T. The key to the hardness argument is the construction of a term $gen : \mathsf{unit} \to \mathsf{int} \vdash \mathbb{C}_{\mathcal{A}} : (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}$ for a given tree automaton $\mathcal{A}$ such that $\mathsf{comp}([\![gen \vdash \mathbb{C}_{\mathcal{A}}]\!]) = \{q\star\} \cup \{q \star \mathcal{S}(T) \mid T \in \mathcal{T}(\mathcal{A})\}$, where $\mathcal{T}(\mathcal{A})$ is the set of trees accepted by $\mathcal{A}$. To that end we take advantage of the term $\vdash \lambda f.f(); f() : (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}$. Observe that it generates complete plays that are very similar to the plays used to represent trees: they have the form $q \star X$, where $X ::= \epsilon \mid q_0\, q_1\, X\, a_1\, q_1\, X\, a_1\, a_0\, X$. To construct $\mathbb{C}_{\mathcal{A}}$ we can equip the term above with additional code that tracks possible states of $\mathcal{A}$, as the input tree is being traversed. In order to cover all possible tree shapes the free identifier $gen : \mathsf{unit} \to \mathsf{int}$ is used as a label generator.

Alternatively, one could readily adapt the EXPTIME-hardness argument for third-order Idealized Algol [15] to the call-by-value setting. This would yield EXPTIME-hardness of observational equivalence for canonical forms typable as

$gen : \mathsf{unit} \to \mathsf{int}, f : ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit} \vdash \mathbb{C} : \mathsf{unit}$.

Theorem 3. *Observational equivalence of* $\mathsf{RML}_{\text{O-Str}}$*-terms in canonical form is EXPTIME-complete.*

Further Directions Does $\mathsf{RML}_{\text{O-Str}}$ capture all the decidable sequents? (We think not.) It would be interesting to identify (and classify) the decidable type sequents of the following related languages. (i) Call-by-value Idealized Algol [5,13], which can be viewed as a fragment of RML with block-allocated storage. It is known that the order-2 fragment is decidable [13] and not order-5 [10]. (ii) The case of Reduced ML [19] (i.e. RML without mkvar) is significantly more complicated. Recently it was shown that terms-in-contexts of the shape $\cdots, x : \beta \to \beta, \cdots \vdash M : \beta \to \beta$, where $\beta = \mathsf{unit}, \mathsf{int}, \mathsf{int\ ref}$, can be represented with automata over infinite alphabets [14].

Another direction we intend to pursue is to implement the model checking algorithm described, building upon the infrastructure of the call-by-name tool Homer [8].

References

1. Abramsky, S., McCusker, G.: Call-by-value games. In: Nielsen, M. (ed.) CSL 1997. LNCS, vol. 1414. Springer, Heidelberg (1998)
2. Abramsky, S., McCusker, G.: Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In: Algol-like Languages. Birkhäuser, Basel (1997)
3. Ahmed, A., Dreyer, D., Rossberg, A.: State-dependent representation independence. In: POPL (2009)
4. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC (2004)
5. Ghica, D.R.: Regular-language semantics for a call-by-value programming language. In: MFPS (2001)
6. Ghica, D.R., McCusker, G.: The regular-language semantics of second-order Idealized Algol. Theor. Comput. Sci. 309(1-3) (2003)
7. Honda, K., Yoshida, N.: Game theoretic analysis of call-by-value computation. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256. Springer, Heidelberg (1997)
8. Hopkins, D., Ong, C.-H.L.: HOMER: A Higher-Order Observational Equivalence Model checkER. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 654–660. Springer, Heidelberg (2009)
9. Hyland, J.M.E., Ong, C.-H.L.: On full abstraction for PCF: I, II, and III. Inf. Comput. 163(2) (2000)
10. Murawski, A.S.: About the undecidability of program equivalence in finitary languages with state. ACM Transactions on Computational Logic 6(4) (2005)
11. Murawski, A.S.: Functions with local state: regularity and undecidability. Theoretical Computer Science 338(1/3) (2005)
12. Murawski, A.S., Ong, C.-H.L., Walukiewicz, I.: Idealized algol with ground recursion, and DPDA equivalence. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 917–929. Springer, Heidelberg (2005)
13. Murawski, A.S., Tzevelekos, N.: Block structure vs. Scope extrusion: Between innocence and omniscience. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 33–47. Springer, Heidelberg (2010)

14. Murawski, A.S., Tzevelekos, N.: Algorithmic nominal game semantics. In: Barthe, G. (ed.) ESOP 2011. LNCS, vol. 6602, pp. 419–438. Springer, Heidelberg (2011)
15. Murawski, A.S., Walukiewicz, I.: Third-order idealized algol with iteration is decidable. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 202–218. Springer, Heidelberg (2005)
16. Pitts, A.M., Stark, I.D.B.: Operational reasoning for functions with local state. In: Higher Order Operational Techniques in Semantics (1998)
17. Reynolds, J.C.: The essence of Algol. In: de Bakker, J.W., van Vliet, J.C. (eds.) Algorithmic Languages, pp. 345–372. North Holland, Amsterdam (1978)
18. Seidl, H.: Deciding equivalence of finite tree automata. SIAM J. Comput. 19(3) (1990)
19. Stark, I.D.B.: Names and Higher-Order Functions. PhD thesis, Univ. of Cambridge (1995)

Krivine Machines and Higher-Order Schemes

S. Salvati and I. Walukiewicz*

Université de Bordeaux, INRIA, CNRS, LaBRI UMR5800

Abstract. We propose a new approach to analysing higher-order recursive schemes. Many results in the literature use automata models generalising pushdown automata, most notably higher-order pushdown automata with collapse (CPDA). Instead, we propose to use the Krivine machine model. Compared to CPDA, this model is closer to lambda-calculus, and incorporates nicely many invariants of computations, as for example the typing information. The usefulness of the proposed approach is demonstrated with new proofs of two central results in the field: the decidability of the local and global model checking problems for higher-order schemes with respect to the mu-calculus.

1 Introduction

Higher-order recursive schemes were introduced by Damm in [Dam82] as a respelling of λY-calculus. Since higher-order recursive schemes were investigated mainly in the formal language community, the tools developed were by and large inspired by the treatment of pushdown-automata and context-free grammars. Subsequent research has shown that it is very useful to have an automata model characterising schemes. For the class of all schemes, we know only one such model, that is higher-order pushdown automata with collapse [HMOS08]. In this paper we propose another model based on Krivine machines [Kri07, Wan07]. The notion of Krivine machine is actually a standard concept in the lambda-calculus community, and it needs almost no adaptation to treat higher-order schemes. We claim that the proposed model offers a fresh tool to analyse schemes. To substantiate we give new proofs of two central results in the field: decidability of local and global model-checking problems for higher-order schemes with respect to the mu-calculus.

The interest in higher-order schemes has been renewed by the discovery by Knapik et al. [KNU02] of the equivalence of higher-order pushdown automata of order n with schemes of order n satisfying a syntactic constraint called *safety*. The safety condition implicitly appeared in Damm's work. Indeed, Damm considers only the so-called *derived types* in the definition of higher-order schemes. This in itself does not restrict the expressive power of schemes if one permits explicit lambda abstractions [DF80]. But then Damm studies only *applicative* schemes which are equivalent to schemes satisfying the safety condition [dM06]. In recent years, higher-order pushdowns have been extended with *panic* operation to

* This work has been supported by ANR 2010 BLAN 0202 01 FREC.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 162–173, 2011.

handle all level 2 schemes [KNUW05, AdMO05], and with *collapse* operation for schemes of all levels [HMOS08]. Higher-order pushdown automata with collapse are at present the main tool to analyse schemes [HMOS08, BO09, BCOS10].

The model checking problem for schemes with respect to the mu-calculus is to decide if a given formula holds in the root of the tree generated by a given scheme. The problem has proved to be very stimulating, and generated many advances in our understanding of schemes. Its decidability has been shown by Ong [Ong06], but even afterwards the problem continued to drive interesting work. Several different proofs of Ong's result have been proposed [HMOS08, KO09]. In a series of recent papers [CHM+08, BO09, BCOS10] the global version of the problem is considered. In the last citation it is shown that the set of nodes satisfying a given mu-calculus formula is definable in a finitary way.

In this paper, we go several steps back with respect to the usual ways of working with higher-order recursive schemes. First, instead of using Damm's definition of higher-order schemes, we turn to the λY-calculus as the means of generating infinite trees. The Y combinator, or the fixpoint combinator, has first been considered in [CR58] and is at the core of Plotkin's PCF [Plo77]. Second, instead of using higher-order collapsible automata as an abstract machine, we use Krivine abstract machine [Kri07]. This machine is much closer to λ-calculus, it performs standard reductions and comes with typing. These features are hard to overestimate as they allow to use standard techniques to express powerful invariants on the computation. For example, in the main proof presented here, we use standard models of the λY-calculus to express such invariants.

Using these tools, we reprove in a rather succinct way Ong's result. Similarly to a recent proof of Kobayashi and Ong [KO09], our proof gives a reduction to a finite parity game. It seems though that our game is simpler. For example, the paper [BCOS10] on global model checking continues to use collapsible pushdown automata and gives an involved proof by induction on the rank of the stack. On the other hand, we can reuse our game to give a short proof of this result. In particular unlike op cit. we use finite trees to represent positions, and standard automata on finite trees to represent sets of winning positions. In this context we would like to mention a result of Kartzow [Kar10] showing that order-2 collapsible stacks can be encoded as trees in such a way that the set of stacks reachable from the initial configuration is a regular set of trees.

Organization of the paper. In the next section we introduce λY-calculus and Krivine machines. We also define formally the local model checking problem. In the following section we reduce the problem to determining a winner in a game over configurations of the Krivine machine, $\mathcal{K}(\mathcal{A}, M)$. In the next section we define a finite game $\mathcal{G}(\mathcal{A}, M)$. We then show that the same player is winning in the two games. This gives decidability of the local model checking problem. In the following section we reuse this result to obtain the proof for the global model checking problem. All missing proofs can be found in the long version of the paper [SW11].

2 Basic Notions

The set of types $\mathcal{T}$ is constructed from a unique *basic type* 0 using a binary operation $\rightarrow$. Thus 0 is a type and if α, β are types, so is $(\alpha \rightarrow \beta)$. The order of a type is defined by: $order(0) = 1$, and $order(\alpha \rightarrow \beta) = max(1 + order(\alpha), order(\beta))$.

A *signature*, denoted Σ, is a set of typed constants, that is symbols with associated types from $\mathcal{T}$. We will assume that for every type $\alpha \in \mathcal{T}$ we have ω^α and $Y^{(\alpha\rightarrow\alpha)\rightarrow\alpha}$ standing for the undefined value and the fixpoint operator. For simplicity of notation we assume that all other constants are of type $0 \rightarrow 0 \rightarrow 0$. In general, in recursive schemes all constants of order 0 and 1 are allowed; it is straightforward to extend our arguments to all such constants.

The set of *simply typed λ-terms* is defined inductively as follows. A constant of type α is a term of type α. For each type α there is a countable set of variables $x^\alpha, y^\alpha, \ldots$ that are also terms of type α. If M is a term of type β and x^α a variable of type α then $\lambda x^\alpha.M$ is a term of type $\alpha \rightarrow \beta$. Finally, if M is of type $\alpha \rightarrow \beta$ and N is of type α then MN is a term of type β. Together with the usual operational semantics of λ-calculus, that is β-reduction, we use δ-reduction ($\rightarrow_\delta$) giving the semantics to the fixpoint operator: $YM \rightarrow_\delta M(YM)$. Thus, the operational semantics of the λY-calculus is the $\beta\delta$-reduction, it is well-known that this semantics is confluent and enjoys subject reduction (*i.e.* the type of terms is invariant under computation). In this paper, we only consider terms in η-long form [Hue76]. This makes the presentation easier as the structure of types is reflected syntactically in terms. Later we will point out the place where we use this assumption. We will also often omit type annotations.

A *Böhm tree* is an unranked ordered, and potentially infinite tree with nodes labelled by ω^α, or terms of the form $\lambda x_1 \ldots x_n.N$; where N is a variable or a constant, and the sequence of lambda abstractions is optional. So for example x^0, $\lambda x^0.\omega^0$ are labels, but $\lambda y^0.x^{0\rightarrow 0}y^0$ is not. A Böhm tree of a term M is obtained as follows. If $M \rightarrow^*_{\beta\delta} \lambda \boldsymbol{x}.N_0N_1 \ldots N_k$ with N_0 a variable or a constant then the root of $BT(M)$ is labelled by $\lambda \boldsymbol{x}.N_0$ and has $BT(N_1), \ldots, BT(N_k)$ as a sequence of its children. If M is not solvable then $BT(M) = \omega^\alpha$, where α is the type of M. If M is of type 0 then given our assumption on the type of constants we get that $BT(M)$ is a binary tree with finite branches ending in ω^0. It is known that λY-calculus allows to define the same trees as recursive schemes [DF80].

Krivine machine. A Krivine machine [Kri07], is an abstract machine that computes the weak head normal form of a λ-term, using explicit substitutions, called *environments*. Environments are functions assigning *closures* to variables, and closures themselves are pairs consisting of a term and an environment. This mutually recursive definition is schematically represented by the grammar:

$$C ::= (M, \rho) \qquad \rho ::= \emptyset \mid \rho[x \mapsto C]$$

As in this grammar, we will use $\emptyset$ for the empty environment. We require that in a closure (M, ρ), the environment is defined for every free variable of M. Intuitively such a closure denotes closed λ-term: it is obtained by substituting for every free variable x of M the lambda term denoted by the closure $\rho(x)$.

A configuration of the Krivine machine is a triple (M, ρ, S), where M is a term, ρ is an *environment*, and S is a *stack* (a sequence of closures with the topmost element on the left). The rules of the Krivine machine are as follows:

$$\begin{aligned}
(\lambda x.M, \rho, (N, \rho')S) &\to (M, \rho[x \mapsto (N, \rho')], S) \\
(YM, \rho, S) &\to (M(YM), \rho, S) \\
(MN, \rho, S) &\to (M, \rho, (N, \rho)S) \\
(x, \rho, S) &\to (M, \rho', S) \qquad \text{where } (M, \rho') = \rho(x)
\end{aligned}$$

Note that the machine is deterministic. We will be only interested in configurations accessible from $(M, \emptyset, \varepsilon)$ for some term M of in η-long form and of type 0. Every such configuration (N, ρ, S) enjoys very strong typing invariants. Environment ρ associates to a variable x^α a closure (K, ρ') so that K has type α; we will say that the closure is of type α too. If N has type $\alpha_1 \to \cdots \to \alpha_n \to 0$, then S is a stack of n closures, with i-th closure from the top being of type α_i.

For aesthetic reasons we prefer to stop the Krivine machine in configurations of the form $(bN_0N_1, \rho, \varepsilon)$, where b is a constant; since b is of type $0 \to 0 \to 0$, the stack must be empty. We shall write this configuration as $(b(N_0, N_1), \rho, \varepsilon)$ to make a link with the Böhm tree being constructed. (Notice that formally from such a configuration the machine should perform two more reductions to put the arguments on the stack.) Thus, if we start with a closed term M of type 0 we get a sequence of reductions from $(M, \emptyset, \varepsilon)$ that is either infinite or terminates in a configuration of a form $(b(N_0, N_1), \rho, \varepsilon)$, thanks to the fact that M is supposed to be in η-long form. At that point we create a node labelled b and start reducing both (N_0, ρ, ε) and (N_1, ρ, ε), that are both in η-long form as well. This process gives at the end a tree labelled with constants that is precisely $BT(M)$; that is the object of our study. Notice that if (N, ρ, S) is reachable from $(M, \emptyset, \varepsilon)$ then N, and the terms that occur in ρ and in S are all subterms of M. One should be careful with a definition of a subterm though. Since we have a fixpoint operator we consider that $N(YN)$ is a subterm of YN. Of course even with this twist, the number of subterms of a term remains finite.

We present an execution of a Krivine machine on an example taken from [KO09]. For clarity, in this example we suspend our convention on types of the constants and take constants $a : 0 \to 0 \to 0$, $b : 0 \to 0$ and $c : 0$. The scheme is defined by $S \mapsto F\,c$ and $F \mapsto \lambda x.a\,x\,(F(b\,x))$ which can be represented by the following term in the λY-calculus:

$$YM\,c \qquad \text{where } M = \lambda f x.a\,x\,(f(b\,x)).$$

Starting from a configuration $(YMc, \emptyset, \varepsilon)$, the Krivine machine produces the following sequence of reductions

$$\begin{aligned}
(YMc, \emptyset, \varepsilon) \to (YM, \emptyset, (c, \emptyset)) \to (M(YM), \emptyset, (c, \emptyset)) \to (M, \emptyset, (YM, \emptyset)(c, \emptyset)) \to \\
(\lambda x.a\,x\,(f(b\,x)), [f \mapsto (YM, \emptyset)], (c, \emptyset)) \to \\
(a\,x\,(f(b\,x)), [f \mapsto (YM, \emptyset)][x \mapsto (c, \emptyset)], \varepsilon)
\end{aligned}$$

At this point we have reached a final configuration and we get the constant a that is the symbol of the root of $BT(YMc)$. We can start reducing separately the two arguments of a, that is reducing the configurations:

$$(x, [f \mapsto (YM, \emptyset)][x \mapsto (c, \emptyset)], \varepsilon) \text{ and } (f(b\,x), [f \mapsto (YM, \emptyset)][x \mapsto (c, \emptyset)], \varepsilon).$$

Parity automata and the definition of the problem. Recall that Σ is a fixed set of constants of type $0 \to 0 \to 0$. These constants label nodes in $BT(M)$. Since $BT(M)$ is an infinite binary tree we can use standard non-deterministic parity automata to describe its properties. Such an automaton has the form

$$\mathcal{A} = \langle Q, \Sigma, q^0 \in Q, \delta : Q \times \Sigma \to \mathcal{P}(Q^2), \Omega : Q \to \{1, \ldots, d\}\rangle \tag{1}$$

where Q is a finite set of states, q^0 is the initial state, δ is the transition function, and Ω is a function assigning a rank (a number between 1 and d) to every state.

In general, an infinite binary tree is a function $t : \{0,1\}^* \to \Sigma$. A run of $\mathcal{A}$ on t is a function $r : \{0,1\}^* \to Q$ such that $r(\varepsilon) = q^0$ and for every sequence $w \in \{0,1\}^*$: $(r(w0), r(w1)) \in \delta(q, t(w))$. The run is accepting if for every infinite path in the tree, the sequence of states assigned to this path satisfies the parity condition determined by Ω; this means that the maximal rank of a state seen infinitely often should be even.

Formally, it may be the case that $BT(M)$ contains also nodes labelled with ω^0. We will simply assume that every tree containing ω^0 is rejected by the automaton. This is frequently done in this context. Handling ω^0 would not be difficult but would require to add one more case in all the constructions. The other, more difficult, solution is to convert a term to a term not generating ω^0.

Definition 1. The (local) model checking problem *is to decide if* $\mathcal{A}$ *accepts* $BT(M)$ *for given* $\mathcal{A}$ *and* M.

3 Game Over Configurations of the Krivine Machine

In this section we will reduce the model checking problem to the problem of determining a winner in a specially constructed parity game.

Given an automaton $\mathcal{A}$ as in (1) we construct the tree of all its possible runs on $BT(M)$. We define the tree of runs formally as we will make one twist to the rules of the Krivine machine. The twist is that in the environment the value of the variable will not be a closure, that is a pair (term, environment), but a triple containing additionally the node of the tree where the closure has been created. For a given M and $\mathcal{A}$ we define the tree of runs $RT(\mathcal{A}, M)$ of $\mathcal{A}$ on $BT(M)$:

- The root of the tree is labelled with $q^0 : (M, \emptyset, \varepsilon)$.
- A node labelled $q : (a(N_0, N_1), \rho, \varepsilon)$ has a successor $(q_0, q_1) : (a(N_0, N_1), \rho, \varepsilon)$ for every $(q_0, q_1) \in \delta(q, a)$.
- A node labelled $(q_0, q_1) : (a(N_0, N_1), \rho, \varepsilon)$ has two successors $q_0 : (N_0, \rho, \varepsilon)$ and $q_1 : (N_1, \rho, \varepsilon)$.
- A node labelled $q : (\lambda x.N, \rho, (v', N', \rho')S)$ has a unique successor labelled $q : (N, \rho[x \mapsto (v', N', \rho')], S)$.

- A node $q : (YN, \rho, S)$ has a unique successor $q : (N(YN), \rho, S)$.
- A node v labelled $q : (NK, \rho, S)$ has a unique successor $q : (N, \rho, (v, K, \rho)S)$. (Here the v closure is created.)
- A node v labelled $q : (x, \rho, S)$ with $\rho(x) = (v', N, \rho')$ has a unique successor labelled $q : (N, \rho', S)$. (We say that the node v uses the v' closure.)

The definition is as expected but for the fact that in the rule for the application we store the current node in the closure that is pushed on the stack. When we use the closure in the variable rule, the stored node does not influence the result, but allows us to detect the exact closure that we are using. This will be important in the proof.

Definition 2. *We use the tree $RT(\mathcal{A}, M)$ to define a game between two players: Eve chooses a successor in nodes of the form $q : (a(N_0, N_1), \rho, \varepsilon)$, and Adam in nodes $(q_0, q_1) : (a(N_0, N_1), \rho, \varepsilon)$. We set the rank of nodes labelled $q : (a(N_0, N_1), \rho, \varepsilon)$ to $\Omega(q)$ and the ranks of all the other nodes to 1. We can use max parity condition to decide who wins an infinite play. Let us call the resulting game $\mathcal{K}(\mathcal{A}, M)$.*

Directly from the definition it follows that Eve has a strategy from the root position in $\mathcal{K}(\mathcal{A}, M)$ iff $\mathcal{A}$ accepts $BT(M)$. The only interesting point to observe is that it is important to disallow rank 0 in the definition of the parity automaton since we assign rank 1 to all "intermediate" positions. This is linked to our handling of infinite sequences of reductions of the Krivine machine that do not reach a head normal form. Such a sequence results in a node labelled ω^0 in the Böhm tree, hence the tree should not be accepted by the automaton. Indeed, in the game $\mathcal{K}(\mathcal{A}, M)$ this will give an infinite sequence of states of rank 1.

Summarizing, the model checking problem is equivalent to deciding who has a winning strategy from the root of $\mathcal{K}(\mathcal{A}, M)$. We will show decidability of the latter problem by reducing the game to a finite game.

4 Finite Game $G(\mathcal{A}, M)$

The game $\mathcal{K}(\mathcal{A}, M)$ may have infinitely many positions as there may be infinitely many closures that are created. In order to obtain a finite game we abstract these closures to some finite set. Closures are created by the application rule, so this is where we will concentrate our efforts. As in the construction for a pushdown game [Wal01] we will use alternation to "disarm" the application rule. Instead of putting a closure on the stack, Eve will make an assumption on the context in which the closure will be used. Adam will be then given a chance to either contest this assumption or to check what happens with the closure when it is used under the assumptions Eve has made. Since the closure can be of higher type, the assumptions are a bit more complicated than in a pushdown game.

Definition 3 (Residuals). *Recall that Q is the set of states of $\mathcal{A}$ and d is the maximal value of the rank function of $\mathcal{A}$. Let $[d]$ stand for the set $\{1, \ldots, d\}$. For every type $\tau = \tau_1 \to \cdots \to \tau_k \to 0$ the set of residuals D_τ is the set of functions $D_{\tau_1} \to \cdots \to D_{\tau_k} \to \mathcal{P}(Q \times [d])$.*

For example, we have that D_0 is $\mathcal{P}(Q\times[d])$ and $D_{0\to 0}$ is $\mathcal{P}(Q\times[d]) \to \mathcal{P}(Q\times[d])$. The meaning of residuals will become clearer when we define the game.

A position of the game $G(\mathcal{A}, M)$ will be of one of the forms:

$$q : (N, \rho, S) \quad \text{or} \quad (q_0, q_1) : (N, \rho, S) \quad \text{or} \quad (q, R) : (N, \rho, S).$$

where q, q_0, q_1 are states of $\mathcal{A}$, N is a term (more precisely a subterm of M), ρ is an environment assigning a residual to every variable that has a free occurrence in N, R is a residual, and S is a stack of residuals. Of course the types of residuals will agree with the types of variables/arguments they are assigned too. As there are only finitely many residuals of each type, the game $G(\mathcal{A}, M)$ has finitely many positions.

We need one more operation before defining the game. Take a rank r and a residual $R : D_{\tau_1} \to \cdots \to D_{\tau_k} \to D_0$. Recall that $D_0 = \mathcal{P}(Q \times [d])$. We define $R\restriction_r$ to be the function such that for every sequence of arguments S:

$$R\restriction_r (S) = \{(q_1, r_1) \in R(S) : r_1 > r\} \cup \{(q_1, r_2) : (q_1, r_1) \in R(S), r_2 \leq r_1 = r\}$$

Intuitively, $(q_1, r_1) \in R(S)$ means that Eve is allowed to reach a leaf labelled with a state q_1 if r_1 is the maximal rank between the creation and the use of the closure. Now suppose that with this residual at hand we see rank r. If $(q_1, r_1) \in R(S)$ and $r_1 > r$ then we are still waiting for r_1 so we just keep the pair. If $r_1 < r$ then such a pair is impossible and is removed. If $r_1 = r$ then in the future we can see any rank not bigger than r. This explains the second component of the sum. If ρ is an environment then $\rho\restriction_r$ is an environment such that for every x: $(\rho\restriction_r)(x) = \rho(x)\restriction_r$.

We have all ingredients to define transitions of the game $G(\mathcal{A}, M)$. Most of the rules are just reformulation of the rules in $\mathcal{K}(\mathcal{A}, M)$:

$$\begin{aligned}
q : (\lambda x.N, \rho, R \cdot S) &\to q : (N, \rho[x \mapsto R], S) \\
q : (a(N_0, N_1), \rho, \varepsilon) &\to (q_0, q_1) : (a(N_0, N_1), \rho, \varepsilon) \qquad \text{for } (q_0, q_1) \in \delta(q, a) \\
(q_0, q_1) : (a(N_0, N_1), \rho, \varepsilon) &\to q_i : (N_i, \rho\restriction_{\Omega(q_i)}, \varepsilon) \qquad \text{for } i = 0, 1 \\
q : (YN, \rho, S) &\to q : (N(YN), \rho, S)
\end{aligned}$$

We now proceed to the rule for application. Consider $q : (NK, \rho, S)$ with K of type $\tau = \tau_1 \to \cdots \to \tau_l \to 0$. We have a transition

$$q : (NK, \rho, S) \to (q, R) : (NK, \rho, S)$$

for every residual $R : D_{\tau_1} \to \ldots D_{\tau_l} \to D_0$. From this position we have transitions

$$\begin{aligned}
&(q, R) : (NK, \rho, S) \to q : (N, \rho, R\restriction_{\Omega(q)} \cdot S) \\
&(q, R) : (NK, \rho, S) \to q' : (K, \rho\restriction_{r'}, R_1 \cdots R_l) \text{ for every } R_1 \in D_{\tau_1}, \ldots, R_l \in D_{\tau_l} \\
&\qquad\qquad \text{and } (q', r') \in R\restriction_{\Omega(q)} (R_1, \ldots, R_l).
\end{aligned}$$

Here $R\restriction_{\Omega(q)}$ is needed to "normalise" the residual, so that it satisfies the invariant described below.

Since we are defining a game we need to say who makes a choice in which vertices. Eve chooses a successor from vertices of the form $q : (NK, \rho, S)$ and $q : (a(N_0, N_1), \rho, S)$. It means that she can choose a residual, and a transition of the automaton. This leaves for Adam the choices in nodes of the form $(q_0, q_1) : (a(N_0, N_1), \rho, S)$ and $(q, R) : (NK, \rho, S)$. So he chooses a direction, or decides whether to accept (by choosing a transition of the first type) or contest the residual proposed by Eve.

Observe that we do not have a rule for nodes with a term being a variable. This means that such a node has no successors, so we need to say who is the winner when the node is reached. Consider a node

$$q : (x, \rho, S) \quad \text{with } \rho(x) = R_x \text{ and } S = R_1 \cdots R_k.$$

Eve wins in this position if $(q, \Omega(q)) \in R_x(R_1, \ldots, R_k)$.

Finally, we define ranks. It will be much simpler to define ranks on transitions instead of nodes. All the transitions will have rank 1 but for two cases: (i) a transition $(q, R) : (NK, \rho, S) \to q' : (K, \rho \restriction_{r'}, R_1 \cdots R_k)$ has rank r'; (ii) a transition $(q_0, q_1) : (a(N_0, N_1), \rho, S) \to q_i : (N_i, \rho \restriction_{\Omega(q_i)}, S \restriction_{\Omega(q_i)})$ has rank $\Omega(q_i)$.

A play is winning for Eve iff the sequence of ranks on transitions satisfies the parity condition: the maximal rank appearing infinitely often is even.

5 Equivalence of $\mathcal{K}(\mathcal{A}, M)$ and $G(\mathcal{A}, M)$

In this section we present the main technical result of the paper

Theorem 1. *Eve wins in $G(\mathcal{A}, M)$ iff Eve wins in $\mathcal{K}(\mathcal{A}, M)$.*

Since $G(\mathcal{A}, M)$ is finite, this gives the decidability of the winner in $\mathcal{K}(\mathcal{A}, M)$ and hence also of the model-checking problem. We will show how to construct the winning strategy for Eve in $G(\mathcal{A}, M)$ from her winning strategy in $\mathcal{K}(\mathcal{A}, M)$. Due to lack of space we omit a dual construction of a winning strategy for Adam in $G(\mathcal{A}, M)$ from his winning strategy in $\mathcal{K}(\mathcal{A}, M)$.

Let us fix a winning strategy σ of Eve in $\mathcal{K}(\mathcal{A}, M)$, and consider the tree $\mathcal{K}_\sigma$ of plays respecting this strategy. This is a subtree of $\mathcal{K}(\mathcal{A}, M)$. We will define the strategy for Eve in $G(\mathcal{A}, M)$ that will use σ to guess residuals in the application rule. The first step before constructing the strategy is to calculate residuals $R(v)$ and $res(v, v')$ for all nodes in the tree $\mathcal{K}_\sigma$.

Residuals $R(v)$ and $res(v, v')$. The crucial step in the proof is the assignment of residuals to positions of $\mathcal{K}(\mathcal{A}, M)$. Thanks to typing, this can be done by induction on the order of type. We will assign a residual $R(v)$ to every v closure. Before proceeding we will need two simple abbreviations. If v is an ancestor of v_1 in $\mathcal{K}_\sigma$ then we write $\max(v, v_1)$ for the maximal rank appearing on the path between v and v_1, including both ends. We write $res(v, v_1)$ for $R(v) \restriction_{\max(v,v_1)}$; intuitively it is residual $R(v)$ as seen from v_1.

Consider an application node v in $\mathcal{K}(\mathcal{A}, M)$. It means that v has a label of the form $q : (NK, \rho, S)$, and its unique successor has the label $q : (N, \rho, (v, K, \rho)S)$. That is the closure (v, K, ρ) is created in v. We will look at all the places where this closure is used and summarize the information about them in $R(v)$.

When the closure is of type 0 then the residual $R(v)$ is a subset of $Q \times [d]$. For every node v' in $\mathcal{K}_\sigma$ labelled $q' : (x, \rho', \varepsilon)$ such that $\rho'(x) = (v, K, \rho)$ we put

$$(q', \max(v, v')) \in R(v)$$

When the closure is of type $\tau_1 \to \cdots \to \tau_k \to 0$, then by induction we assume that we have residuals for all closures of types $\tau_1, \ldots, \tau_k$. This time $R(v) : D_{\tau_1} \to \ldots D_{\tau_k} \to \mathcal{P}(Q \times [d])$. Take a node v' using the closure. Its label has the form $q' : (x, \rho', S')$ for some x, ρ' and S' such that $\rho'(x) = (v, K, \rho)$. The stack S' has the form $(v_1, N_1, \rho_1) \ldots (v_k, N_k, \rho_k)$ with N_i of type τ_i. We put

$$(q', \max(v, v')) \in R(res(v_1, v'), \ldots, res(v_k, v')) \ . \tag{2}$$

We extend the definition of residuals to closures, environments and stacks. For a closure (v, K, ρ) we define $res((v, K, \rho), v') = res(v, v')$. For an environment ρ, the environment $\rho' = res(\rho, v')$ is obtained by setting $\rho'(x) = res(\rho(x), v')$ for every variable x. Similarly, $res(S, v')$ is S where $res(\cdot, v')$ is applied to every element of the stack. With this notation the condition (2) can be rewritten as $(q', \max(v, v')) \in R(res(S', v'))$.

The strategy in $G(\mathcal{A}, M)$ Now we are ready to define the strategy for Eve in $G(\mathcal{A}, M)$. It will use positions in the game $\mathcal{K}(\mathcal{A}, M)$ and the strategy σ as hints. The new strategy will take a pair of positions (v_1, v_2) with v_1 in $G(\mathcal{A}, M)$, and v_2 in $\mathcal{K}(\mathcal{A}, M)$. It will then give a new pair of positions (v_1', v_2') such that v_1' is a successor v_1, and v_2' is reachable from v_2 using the strategy σ. Moreover, all visited pairs (v_1, v_2) will satisfy the following invariant:

$$\begin{gathered} v_1 \text{ is labelled by } q : (N, \rho_1, S_1) \text{ and } v_2 \text{ is labelled by } q : (N, \rho_2, S_2); \\ \text{where } \rho_1 = res(\rho_2, v_2) \text{ and } S_1 = res(S_2, v_2). \end{gathered}$$

The initial positions in both games have the same label $q^0 : (M, \varepsilon, \emptyset)$, so the invariant is satisfied. In order to define the strategy we will consider one by one the rules defining the transitions in $G(\mathcal{A}, M)$.

In most of the cases the strategy in $G(\mathcal{A}, M)$ just copies the moves of the strategy in $\mathcal{K}(\mathcal{A}, M)$. The only complicated case is the application rule.

Suppose that the term in the label of v_1 is an application, say $q : (NK, \rho_1, S_1)$. By our invariant we have a position v_2 labelled by $q : (NK, \rho_2, S_2)$, where $\rho_1 = res(\rho_2, v_2)$ and $S_1 = res(S_2, v_2)$. The strategy in $G(\mathcal{A}, M)$ is to choose $R(v_2)$, that is to go from v_1 to the node v_1' labelled $(q, R(v_2)) : (NK, \rho_1, S_1)$. From this node Adam can choose either

$$q : (N, \rho_1, (R(v_2) \restriction_{\Omega(q)}) \cdot S_1), \quad \text{or} \tag{3}$$

$$q' : (K, \rho_1 \restriction_{r'}, R_1 \ldots R_l) \quad \text{where } (q', r') \in R(v_2) \restriction_{\Omega(q)} (R_1, \ldots, R_l). \tag{4}$$

Suppose Adam chooses (3). By definition $R(v_2) \restriction_{\Omega(q)} = res(v_2, v_2)$. Hence the stack $(R(v_2) \restriction_{\Omega(q)}) \cdot S_1$ is just $res((v_2, K, \rho_2)S_2, v_2)$. The unique successor v_2' of v_2 is labelled by $q : (N, \rho_2, (v_2, K, \rho_2)S_2)$. So the pair (v_1', v_2') satisfies the invariant.

Let us now examine the case when Adam chooses a node of the form (4). By definition of $R(v_2)$ this means that in $\mathcal{K}_\sigma$ there is a node v_2' labelled $q' : (x, \rho_2', S_2')$ with $\rho_2'(x) = (v_2, K, \rho_2)$ and $res(S_2', v_2') = R_1 \dots R_l$. Moreover $r' = \max(v_2, v_2')$. The successor v_2'' of v_2' is labelled by $q' : (K, \rho_2, S_2')$. We can take it as a companion for v_1' since by easy calculation $\rho_1 \restriction_{r'} = res(\rho_2, v_2) \restriction_{\max(v_2, v_2'')} = res(\rho_2, v_2'')$. Hence the strategy is able to preserve the invariant.

It is easy to check that the above strategy is winning. It remains to check what happens when a maximal play is finite. This means that the path ends in a pair (v_1, v_2) where v_1 is a variable node. Such a node is labelled by $q : (x, \rho_1, S_1)$. To show that this position is winning requires a small calculation proving that $(q, \Omega(q)) \in R_x(S_1)$ for $R_x = \rho_1(x)$.

6 Global Model Checking

The finite game $\mathcal{G}(\mathcal{A}, M)$ allows us also to compute a finite representation of the set of winning positions of Eve in the game $\mathcal{K}(\mathcal{A}, M)$. For this we represent positions in the later game as trees and show that the set of winning positions for Eve is regular: the tree representations of winning positions are recognizable by a finite tree automaton.

Recall that positions of $\mathcal{K}(\mathcal{A}, M)$ are of the form $q : (N, \rho, S)$ where N is a subterm of M, ρ is an environment assigning a closure to every free variable of N, and S is a stack of closures. Recall also that terms from all the closures of ρ and S are subterms of M.

We start by defining a representation of closures as trees. We take the set of all subterms of M as the alphabet: the arity of a letter N being the number of free variables in N. So, for example, if N does not have free variables then a node labelled by N is a leaf in a tree. When N has free variables $x_1, \dots, x_l$; a closure (N, ρ) is represented by a tree whose root is labelled by N and the subtree t_i rooted in i-th child represents $\rho(x_i)$; for $i = 1, \dots, l$. For t of this form, we write $term(t)$ to denote the lambda term obtained by substituting $term(x_i)$ for x_i in N, for $i = 1, \dots, l$. Observe that $term(t)$ is closed: it has no free variables.

A position $q : (N, \rho, S)$ of $\mathcal{K}(\mathcal{A}, M)$ is represented as a tree whose root labelled $q : \tau_N$ has the sequence of children: the tree rooted in the first child representing (N, ρ), and the others representing the closures from S in the same order as in S. Hence the number of children of the root depends on the size of S that in turn is determined by the type τ_N of N.

Since representations of configurations are finite trees over a finite ranked alphabet, we can use standard finite automata to recognize sets of such trees. This gives a notion of a regular set of positions of $\mathcal{K}(\mathcal{A}, M)$.

Theorem 2. *For every $\mathcal{A}$ and M: the set of representations of positions of $\mathcal{K}(\mathcal{A}, M)$ that are winning for Eve is regular.*

The proof uses reduction to finite games. Let $term(N, \rho, S)$ be the term denoted by the closure (N, ρ) applied to terms denoted by the closures in S. It is a closed term of type 0. Of course the behaviours of the Krivine machine from

(N, ρ, S) and $(term(N, \rho, S), \emptyset, \varepsilon)$ are the same, that is they give the same Böhm trees. This implies that Eve wins from $q : (N, \rho, S)$ in $K(\mathcal{A}, M)$ iff she wins from $q : (term(N, \rho, S), \emptyset, \varepsilon)$ in $K(\mathcal{A}, term(N, \rho, S))$. By the reduction theorem (Theorem 1) the later is equivalent to Eve winning from $q : (term(N, \rho, S), \emptyset, \varepsilon)$ in the finite game $G(\mathcal{A}, term(N, \rho, S))$. This condition can be checked by a finite alternating automaton that essentially applies the rules of the construction of the finite game.

7 Conclusions

We have proposed to use Krivine machines to analyse higher-order recursive schemes. The rich structure of this formalism allows to write compact and powerful invariants on computation giving a rather succinct proof of decidability of local model checking. The result on global model checking shows that the structure of configurations of the Krivine machine, although rich, is quite easy to work with.

The proof of Kobayashi and Ong [KO09] is a remarkable achievement showing that one can prove the result with the assumption method (in the spirit of [Wal01]) on the level of terms instead of higher-order pushdown automata with collapse (CPDA). Our residuals are very similar to the additional indices in types introduced in that paper. Also handling of ranks via $\downarrow_{\Omega(q)}$ operation is similar in both proofs. The typing rule for application gives naturally essentially the same rule as we use here. The finite game in that paper is rather different though, as the typing system of Kobayashi and Ong has not been designed to handle fixpoints or lambda-abstraction. The proof of the correctness of the reduction is just different since without configurations of the Krivine machine it is very difficult to state the correspondence between nodes in the tree generated by the scheme and nodes in the finite game.

In his original proof Ong [Ong06] constructed an infinite term, called computation tree, and then showed how to find the Böhm tree inside it. So naturally he concentrated on lambda-abstractions and variables. To find the paths of the Böhm tree he needed a fine description of computation offered by game semantics. Here, thanks to our invariants expressed in terms of closures, we manage to avoid the problem of finding paths of the Böhm tree inside the computation tree. This is probably the biggest technical simplification of our approach.

In the present paper we have kept models of λY-calculus in the background. Yet, the two proofs strongly suggest that there is a finitary model where we can calculate the behaviour of a fixed automaton on a given term. It would be very interesting to find a useful representation of this model.

References

[AdMO05] Aehlig, K., de Miranda, J.G., Ong, C.-H.L.: Safety is not a restriction at level 2 for string languages. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 490–504. Springer, Heidelberg (2005)

[BCOS10] Broadbent, C., Carayol, A., Ong, L., Serre, O.: Recursion schemes and logical reflection. In: LICS, pp. 120–129 (2010)

[BO09] Broadbent, C., Ong, C.-H.L.: On global model checking trees generated by higher-order recursion schemes. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 107–121. Springer, Heidelberg (2009)

[CHM+08] Carayol, A., Hague, M., Meyer, A., Ong, L., Serre, O.: Winning regions of higher-order pushdown games. In: LICS, Pittsburgh, United States, pp. 193–204 (2008)

[CR58] Curry, H.B., Feys, R.: Combinatory Logic, vol. 1. North-Holland Publishing Co., Amsterdam (1958)

[Dam82] Damm, W.: The IO- and OI-hierarchies. Theoretical Computer Science 20, 95–207 (1982)

[DF80] Damm, W., Fehr, E.: A schematalogical approach to the alalysis of the procedure concept in Algol-languages. In: CLAAP, vol. 1, pp. 130–134. Université de Lille (1980)

[dM06] de Miranda, J.: Structures generated by Higher-Order Grammars and the Safety Constraint. PhD thesis, Oxford University (2006)

[HMOS08] Hague, M., Murawski, A.S., Ong, C.-H.L., Serre, O.: Collapsible pushdown automata and recursion schemes. In: LICS, pp. 452–461 (2008)

[Hue76] Huet, G.: Résolution d'équations dans des langages d'ordre 1,2,.,ω. Thèse de doctorat en sciences mathématiques, Université Paris VII (1976)

[Kar10] Kartzow, A.: Collapsible pushdown graphs of level 2 are tree-automatic. In: STACS. LIPIcs, vol. 5, pp. 501–512 (2010)

[KNU02] Knapik, T., Niwinski, D., Urzyczyn, P.: Higher-order pushdown trees are easy. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 205–222. Springer, Heidelberg (2002)

[KNUW05] Knapik, T., Niwinski, D., Urzycyzn, P., Walukiewicz, I.: Unsafe grammars and pannic automata. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1450–1461. Springer, Heidelberg (2005)

[KO09] Kobayashi, N., Ong, L.: A type system equivalent to modal mu-calculus model checking of recursion schemes. In: LICS, pp. 179–188 (2009)

[Kri07] Krivine, J.-L.: A call-by-name lambda-calculus machine. Higher-Order and Symbolic Computation 20(3), 199–207 (2007)

[Ong06] Luke Ong, C.-H.: On model-checking trees generated by higher-order recursion schemes. In: LICS, pp. 81–90 (2006)

[Plo77] Plotkin, G.D.: LCF considered as a programming language. Theor. Comput. Sci. 5(3), 223–255 (1977)

[SW11] Salvati, S., Walukiewicz, I.: Krivine machines and higher-order schemes (2011), http://www.labri.fr/perso/salvati/downloads/articles/hpda-dec.pdf

[Wal01] Walukiewicz, I.: Pushdown processes: Games and model checking. Information and Computation 164(2), 234–263 (2001)

[Wan07] Wand, M.: On the correctness of the Krivine machine. Higher-Order and Symbolic Computation 20, 231–235 (2007), 10.1007/s10990-007-9019-8

Relating Computational Effects by $\top\top$-Lifting

Shin-ya Katsumata

Research Institute for Mathematical Sciences
Kyoto University, Kyoto, 606-8502, Japan
sinya@kurims.kyoto-u.ac.jp

Abstract. We consider the problem of establishing a relationship between two interpretations of base type terms of a λ_c-calculus with algebraic operations. We show that the given relationship holds if it satisfies a set of natural conditions. We apply this result to comparing interpretations of new name creation by two monads: Stark's new name creation monad [25] and a global counter monad.

1 Introduction

Suppose that two monadic semantics $\mathcal{A}_1, \mathcal{A}_2$ are given to a call-by-value functional language, and each semantics $\mathcal{A}_i$ interprets a base type b by a set $A_i b$ and computational effects by a monad $\mathcal{T}_i$. After comparing these semantics, you find a relationship $Vb \subseteq A_1 b \times A_2 b$ between base type values, and also a relationship $Cb \subseteq T_1 A_1 b \times T_2 A_2 b$ between base type computations. We then consider the following problem:

> For any well-typed term $x_1 : b_1, \cdots, x_n : b_n \vdash M : b$ and $(v_i, w_i) \in Vb_i$, do we have $(\mathcal{A}_1[\![M]\!](v_1, \cdots, v_n), \mathcal{A}_2[\![M]\!](w_1, \cdots, w_n)) \in Cb$?

We name this problem *effect simulation problem* and tackle it under the situation where 1) the call-by-value functional language is the simply typed λ_c-calculus with products, coproducts, effect-free constants and *algebraic operations* [22], and 2) the underlying category of a semantics is a bi-CCC with a strong monad. We show that the answer of the effect simulation problem is "yes" if I) monad units η_1, η_2 map pairs in V to pairs in C, II) V is closed under effect-free constants and III) C is closed under algebraic operations in the λ_c-calculus. We prove this by extending Mitchell's representation independence proof [17] with logical relations for monads constructed by *categorical* $\top\top$*-lifting [11]*, which is a semantic formulation of the *leapfrog method* introduced by Lindley and Stark [14,15]. The point of this result is the generality: it holds with any monad, algebraic operation and relation V and C. We demonstrate the flexibility of our solution by showing a general comparison theorem of two monadic semantics related by strong monad morphisms (Section 4), and comparing two interpretations of new name creation by Stark's new name creation monad [25] and a global state monad (Section 5). In Section 6 we consider the effect simulation problem under the presence of recursive functions.

Preliminary. A bold letter, such as $\boldsymbol{x}$, abbreviates a sequence $x_1 \cdots x_n$. The length of the sequence is written by $|\boldsymbol{x}|$. We regard every set as a discrete category. We write $\Rightarrow$ and λ for exponentials and currying operators in a CCC. A *bi-CCC* is a CCC with finite coproducts. For a monad (T, η, μ) and a morphism $f : I \to TJ$, we write $f^{\#}$ for $\mu_J \circ Tf$.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 174–185, 2011.

2 The λ_c-Calculus with Algebraic Operations

We adopt the simply-typed λ_c-calculus with effect-free constants and *algebraic operations* [22] as an idealised call-by-value functional language. In Section 6 we add recursive functions.

Let B be the set of base types. We use b (and its variants) to range over B. An effect-free constant in the calculus takes a value of type $b_1 \times \ldots \times b_n$ and returns a value of type $\sum_{i=1}^{m} \prod_{j=1}^{l_i} b'_{ij}$. We encode this type information by an element in $B^* \times (B^*)^*$. Examples of effect-free constants are the equality predicate: $\mathbf{eq}^b : b \times b \to 1 + 1$ and the division function with a reminder and error: $\mathbf{div} : nat \times nat \to nat \times nat + 1$. An algebraic operation in the calculus has the form $o(\boldsymbol{x}_1^{\boldsymbol{b}_1}.M_1, \ldots, \boldsymbol{x}_n^{\boldsymbol{b}_n}.M_n)$. The variables $\boldsymbol{x}_i$ are bound in each subterm M_i. The number of subterms of the algebraic operation and the types of the bound variables in each subterm are specified by an element in $(B^*)^*$ called *arity*. For instance, an algebraic operation of arity $(\epsilon, b_1 b_2)$ looks like $o(M_1, x_1^{b_1} x_2^{b_2}.M_2)$. The symbols of effect-free constants and algebraic operations added to the calculus are specified by a *signature* Σ over B. It assigns a set of symbols to each element in $B^* \times (B^*)^* + (B^*)^*$. We assume that $\Sigma(x)$ is disjoint with each other. We write $\Sigma^{\boldsymbol{b} \to a}$ and Σ^a for the sets $\Sigma(\iota_1(\boldsymbol{b}, a))$ and $\Sigma(\iota_2(a))$, respectively.

We define the computational lambda calculus $\lambda_c(B, \Sigma)$ over the set B of base types and a signature Σ over B. The types and terms of $\lambda_c(B, \Sigma)$ are defined as follows:

$$
\begin{aligned}
\rho &::= b \mid \rho \Rightarrow \rho \mid 1 \mid \rho \times \rho \mid 0 \mid \rho + \rho \\
M &::= x \mid \lambda x^{\rho}.\, M \mid MM \mid * \mid (M, M) \mid \pi_i(M) \mid \bot_\rho \mid \\
&\quad \iota_i(M) \mid \delta(M, x.M, x.M) \mid \mathsf{let}\ x = M\ \mathsf{in}\ M \mid c\ M \mid o_\rho(\boldsymbol{x}^{\boldsymbol{b}}.M, \cdots, \boldsymbol{x}^{\boldsymbol{b}}.M)
\end{aligned}
$$

where c, o ranges over the set of symbols for effect-free constants and algebraic operations specified by Σ. The type system of $\lambda_c(B, \Sigma)$ extends the one for the simply typed lambda calculus with products and sums (see e.g. [18]). The term $\bot_\rho$ denotes the unique term of type $0 \Rightarrow \rho$, and the δ-term denotes the sum elimination. The typing rules for the last three terms are the following:

$$
\frac{\Gamma \vdash M : \rho \quad \Gamma, x : \rho \vdash N : \sigma}{\Gamma \vdash \mathsf{let}\ x = M\ \mathsf{in}\ N : \sigma} \qquad \frac{\Gamma \vdash M : b_1 \times \cdots \times b_n \quad c \in \Sigma^{b_1 \cdots b_n \to (\boldsymbol{b}'_1, \cdots, \boldsymbol{b}'_m)}}{\Gamma \vdash c\ M : \sum_{i=1}^{m} \prod_{j=1}^{|\boldsymbol{b}'_i|} b'_{ij}}
$$

$$
\frac{\Gamma, x_{i1} : b_{i1}, \cdots, x_{i|\boldsymbol{b}_i|} : b_{i|\boldsymbol{b}_i|} \vdash M_i : \rho \quad 1 \le i \le n \quad o \in \Sigma^{(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n)}}{\Gamma \vdash o_\rho(\boldsymbol{x}_1^{\boldsymbol{b}_1}.M_1, \cdots, \boldsymbol{x}_n^{\boldsymbol{b}_n}.M_n) : \rho}\ .
$$

We move to the semantics of the $\lambda_c(B, \Sigma)$-calculus.

Definition 1 **([22]).** *Let $\mathcal{T} = (T, \eta, \mu, \theta)$ be a strong monad over a CCC $\mathbb{C}$ and $Z \in \mathbb{C}$. We write $st^Z_{I,J} : I \times (Z \Rightarrow J) \to Z \Rightarrow (I \times J)$ for the strength of $Z \Rightarrow -$. A Z-ary* algebraic operation *for $\mathcal{T}$ is a natural transformation $\alpha_I : Z \Rightarrow TI \to TI$ such that*

$$
\alpha_{I \times J} \circ Z \Rightarrow \theta_{I,J} \circ st^Z_{I,J} = \theta_{I,J} \circ I \times \alpha_J, \qquad \mu_I \circ \alpha_{TI} = \alpha_I \circ Z \Rightarrow \mu_I.
$$

Let $A : B \to \mathbb{C}$ be a functor to a bi-CCC $\mathbb{C}$. We extend A to a functor $A' : (B^*)^* \to \mathbb{C}$ by $A'(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n) = \sum_{i=1}^{n} \prod_{j=1}^{|\boldsymbol{b}_i|} A b_{ij}$. Below we simply write A for A'.

Definition 2. *A $\lambda_c(B,\Sigma)$-structure is a tuple $\mathcal{A} = (\mathbb{C}, \mathcal{T}, A, k, \alpha)$ where $\mathbb{C}$ is a bi-CCC, $\mathcal{T}$ is a strong monad on $\mathbb{C}$, A is a functor of type $B \to \mathbb{C}$, k assigns to each $c \in \Sigma^{\boldsymbol{b}\to a}$ a morphism $kc : \prod_{i=1}^{|\boldsymbol{b}|} Ab_i \to Aa$, and α assigns to each $o \in \Sigma^a$ an Aa-ary algebraic operation αo for $\mathcal{T}$.*

We write $\mathcal{A}_1 \times \mathcal{A}_2$ to mean the evident product of two $\lambda_c(B,\Sigma)$-structures $\mathcal{A}_1, \mathcal{A}_2$. Each $\lambda_c(B,\Sigma)$-structure $\mathcal{A}$ determines a natural interpretation $\mathcal{A}[\![-]\!]$ of types and well-typed terms of $\lambda_c(B,\Sigma)$ in the category of $\mathcal{A}$ (see e.g. CBV Translation in [1]). Effect-free constants and algebraic operations are interpreted as follows:

$$\mathcal{A}[\![c\ M]\!] = T(kc) \circ \mathcal{A}[\![M]\!]$$
$$\mathcal{A}[\![o_\rho(\boldsymbol{x}_1.M_1, \cdots, \boldsymbol{x}_n.M_n)]\!] = \alpha o_{\mathcal{A}[\![\rho]\!]} \circ \langle \lambda^{|\boldsymbol{x}_1|}(\mathcal{A}[\![M_1]\!]), \cdots, \lambda^{|\boldsymbol{x}_n|}(\mathcal{A}[\![M_n]\!])\rangle.$$

3 Effect Simulation Problem

The main problem we consider is the *effect simulation problem*. We first introduce a set-theoretic version of it. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be $\lambda_c(B,\Sigma)$-structures over **Set**. A *simulation* between $\mathcal{A}_1$ and $\mathcal{A}_2$ is a pair (V, C) where V and C are B-indexed families of binary relations such that $Vb \subseteq \mathcal{A}_1[\![b]\!] \times \mathcal{A}_2[\![b]\!]$ and $Cb \subseteq T_1\mathcal{A}_1[\![b]\!] \times T_2\mathcal{A}_2[\![b]\!]$; we call V and C *value simulation* and *computation simulation*, respectively. The effect simulation problem is the following:

> Suppose that a simulation (V, C) between $\mathcal{A}_1$ and $\mathcal{A}_2$ is given. Then for any well-typed term $x_1 : b_1, \cdots, x_n : b_n \vdash M : b$ and $(v_i, w_i) \in Vb_i$, do we have $(\mathcal{A}_1[\![M]\!](\boldsymbol{v}), \mathcal{A}_2[\![M]\!](\boldsymbol{w})) \in Cb$?

Example 1. Let B be the set of base types and Σ be the signature that specifies only two algebraic operation symbols: $\mathsf{null} \in \Sigma^\epsilon$ and $\mathsf{join} \in \Sigma^{(\epsilon,\epsilon)}$. We regard $\lambda_c(B,\Sigma)$ as a call-by-value functional language with constructors for nondeterministic computation. The standard semantics of $\lambda_c(B,\Sigma)$ is given by the $\lambda_c(B,\Sigma)$-structure $\mathcal{A}_1 = (\mathbf{Set}, \mathcal{T}_p, A, k, \alpha_1)$, where $\mathcal{T}_p$ is the finite powerset monad, and α_1 assigns algebraic operations by $\alpha_1(\mathsf{null}) = \emptyset$ and $\alpha_1(\mathsf{join})(x, y) = x \cup y$. On the other hand, one may represent nondeterministic choices by finite lists instead of finite sets. This representation corresponds to the semantics of $\lambda_c(B,\Sigma)$ by the $\lambda_c(B,\Sigma)$-structure $\mathcal{A}_2 = (\mathbf{Set}, \mathcal{T}_m, A, k, \alpha_2)$ where $\mathcal{T}_m$ is the free monoid monad, and α_2 assigns algebraic operations by $\alpha_2(\mathsf{null}) = \epsilon$ (the empty list) and $\alpha_2(\mathsf{join})(x, y) = x \cdot y$ (the concatenation of two lists).

We expect that for any well-typed term $x_1 : b_1, \cdots, x_n : b_n \vdash M : b$ and $v_i \in Ab_i$, the denotation $\mathcal{A}_1[\![M]\!](\boldsymbol{v})$ gives the set of possible choices listed up in $\mathcal{A}_2[\![M]\!](\boldsymbol{v})$. That is, we expect that the answer to the effect simulation problem with value simulation $Vb = \{(v, v) \mid v \in Ab\}$ and computation simulation $Cb = \{(X, l) \in T_pAb \times T_mAb \mid X =$ the set of elements in $l\}$ is yes.

We move on to the general situation where the underlying categories are other than **Set**. To formulate the concept of relation between two objects from different categories, we first formulate the concept of predicate over objects in arbitrary category in terms of *fibrational category theory*, then derive the concept of relation as predicates over product categories. Formulating logical relations in fibrational category theory is advocated by Hermida [9], which subsumes *subscone* [19]; see Example 3-1.

Here we give brief definitions of fibration and related concepts; see [10] for the complete detail. Let $p : \mathbb{E} \to \mathbb{B}$ be a functor. We say that $X \in \mathbb{E}$ is *above* $I \in \mathbb{B}$ if $pX = I$. We use the same word for morphisms in $\mathbb{E}$ and $\mathbb{B}$. A *fibre category* over $I \in \mathbb{B}$ is the subcategory of $\mathbb{E}$ consisting of objects above $I \in \mathbb{B}$ and morphisms above id_I. We next assume that p is faithful. One easily sees that $\mathbb{E}_I$ is a preorder. In this situation, we regard $\mathbb{E}_I$ as the preorder of predicates on I. For $X, Y \in \mathbb{E}$, by $f : X \dot{\to} Y$ we mean that $f \in \mathbb{B}(pX, pY)$ and there exists a (necessarily unique) morphism $\dot{f} : X \to Y$ above f. We call $\dot{f}$ the *witness* of $f : X \dot{\to} Y$. The statement $f : X \dot{\to} Y$ means that f is a morphism that sends elements satisfying X to those satisfying Y.

Definition 3. *A* partial order bifibration with fibrewise small products *is a faithful functor $p : \mathbb{E} \to \mathbb{B}$ such that:*

(Partial Order) *Each fibre category is a partial order.*
(Fibration) [1] *For any $I \in \mathbb{B}$, $Y \in \mathbb{E}$ and $f \in \mathbb{B}(I, pY)$, there exists $X \in \mathbb{E}$ above I such that $f : X \dot{\to} Y$ and the following property holds: for any $Z \in \mathbb{E}$ and $h : pZ \to I$, $f \circ h : Z \dot{\to} Y$ implies $h : Z \dot{\to} X$. This property and $\mathbb{E}_I$ being a partial order imply that X is unique; hence we write it by f^*Y, and the witness of $f : f^*Y \dot{\to} Y$ by $\overline{f}Y$. Furthermore, for any $f \in \mathbb{B}(I, J)$, the mapping $Y \in \mathbb{E}_J \mapsto f^*Y \in \mathbb{E}_I$ extends to a functor $f^* : \mathbb{E}_J \to \mathbb{E}_I$. We call it* inverse image functor. *Intuitively, f^*Y corresponds to the predicate $\{i \in I \mid f(i) \in Y\}$ on I.*
(Bi–) *Each inverse image functor has a left adjoint called* direct image functor.
(Fibrewise Small Products) *Each fibre category has small products and inverse image functors (necessarily) preserve them.*

Definition 4. *A* category for logical relations *over a bi-CCC $\mathbb{C}$ is a partial order bifibration $p : \mathbb{E} \to \mathbb{C}$ with fibrewise small products*[2] *such that $\mathbb{E}$ is a bi-CCC and p strictly preserves the bi-cartesian closed structure. Notational convention: we write the bi-cartesian closed structure on $\mathbb{E}$ with dots on the top, like $\dot{\Rightarrow}, \dot{1}, \dot{\times}, \dot{0}, \dot{+}, \dot{ev}, \dot{\lambda}, \dot{\langle -, - \rangle}, \cdots$.*

In [10, Section 9.2], it is discussed when a fibration becomes a category for logical relations. Particularly, the subobject fibration of any presheaf category is a category for logical relations. Below we see a special case: the subobject fibration of **Set**.

Example 2. [10, Chapter 0] We define the category **Pred** by the following data: an object in **Pred** is a pair (X, I) where X is a subset of I and a morphism from (X, I) to (Y, J) is a function $f : I \to J$ such that for any $i \in X$, $f(i) \in Y$. This category is equivalent to the category of subobjects of **Set**. The evident forgetful functor $\pi :$ **Pred** $\to$ **Set** is a bifibration; the inverse image functor for a function $f : I \to J$ is given by $f^*(Y, J) = (\{x \mid f(x) \in Y\}, I)$, and it has a left adjoint given by $f_*(X, I) = (\{f(x) \mid x \in X\}, J)$. The fibre category $\mathbf{Pred}_I$ is the poset $(2^I, \subseteq)$, and the intersection gives small products. Therefore π is a partial order bifibration with fibrewise small products.

The category **Pred** has a bi-cartesian closed structure that is strictly preserved by π [10, Exercise 9.2.1]. The exponential is given by $(X, I) \dot{\Rightarrow} (Y, J) = (\{f \mid \forall x \in X \,.\, f(x) \in Y\}, I \Rightarrow J)$. To summarise, π is a category for logical relations.

[1] This definition of fibration exploits the assumption that p is faithful. The concept of fibration is actually defined on arbitrary functor [10, Definition 1.1.3].
[2] We actually only use fibrewise products up to the cardinality of the set B of base types.

Proposition 1. *Let $p : \mathbb{E} \to \mathbb{B}$ be a category for logical relations, $\mathbb{C}$ be a bi-CCC and $F : \mathbb{C} \to \mathbb{B}$ be a finite-product preserving functor. Then the pullback $F^*(p) : F^*(\mathbb{E}) \to \mathbb{C}$ of p along F is a category for logical relations.*

Proof. This is a straightforward generalisation of the proof that any subscone is a CCC [19]. We use direct image functors to construct finite coproducts in $F^*(\mathbb{E})$.

Example 3.
1. A *subscone* [19] over a bi-CCC $\mathbb{C}$ is the category obtained by pulling back $\pi : \mathbf{Pred} \to \mathbf{Set}$ along the global element functor $\mathbb{C}(1, -) : \mathbb{C} \to \mathbf{Set}$. The leg from the subscone to $\mathbb{C}$ is a category for logical relations.
2. We pull-back π along the product functor $- \times - : \mathbf{Set}^2 \to \mathbf{Set}$. This yields the category **Rel** of binary relations, and the leg $q : \mathbf{Rel} \to \mathbf{Set}^2$ is a category for logical relations.

Having abstracted the concept of predicates in terms of fibrational category theory, we now generalise the effect simulation problem to the following *effect property problem*. Let $\mathcal{A} = (\mathbb{C}, \mathcal{T}, A, k, \alpha)$ be a $\lambda_c(B, \Sigma)$-structure and $p : \mathbb{E} \to \mathbb{C}$ be a category for logical relations. A *property* over $\mathcal{A}$ is a pair (V, C) of functors $V, C : B \to \mathbb{E}$ such that for all base types $b \in B$, Vb is above Ab and Cb is above TAb. The problem is:

> Given a property (V, C) over $\mathcal{A}$, does any well-typed term $x_1 : b_1, \cdots, x_n : b_n \vdash M : b$ satisfy $\mathcal{A}[\![M]\!] : Vb_1 \dot{\times} \cdots \dot{\times} Vb_n \dot{\to} Cb$?

An effect simulation problem between two $\lambda_c(B, \Sigma)$-structures $\mathcal{A}_1$ and $\mathcal{A}_2$ is nothing but an effect property problem over $\mathcal{A}_1 \times \mathcal{A}_2$; particularly the set-theoretic one in the beginning of this section uses $q : \mathbf{Rel} \to \mathbf{Set}^2$ as a category for logical relations.

We say that a property (V, C) over a $\lambda_c(B, \Sigma)$-structure $\mathcal{A} = (\mathbb{C}, \mathcal{T}, A, k, \alpha)$ satisfies:

(I) if for all base types $b \in B$, we have $\eta_{Ab} : Vb \dot{\to} Cb$.
(C1) if for all $(\boldsymbol{b}, a) \in B^* \times (B^*)^*$ and effect-free constant symbols $c \in \Sigma^{\boldsymbol{b} \to a}$, we have $kc : \dot{\prod}_{i=1}^{|\boldsymbol{b}|} Vb_i \dot{\to} Va$.
(C2) if for all arities $a \in (B^*)^*$, algebraic operation symbols $o \in \Sigma^a$ and base types $b \in B$, we have $\alpha o_{Ab} : Va \dot{\Rightarrow} Cb \dot{\to} Cb$.

Theorem 1. *Let $\mathcal{A}$ be a $\lambda_c(B, \Sigma)$-structure and (V, C) be a property over $\mathcal{A}$ that satisfies (I), (C1) and (C2). Then for any well-typed term $x_1 : b_1, \cdots, x_n : b_n \vdash M : b$, we have $\mathcal{A}[\![M]\!] : Vb_1 \dot{\times} \cdots \dot{\times} Vb_n \dot{\to} Cb$.*

The rest of this section is the proof of the above theorem. The proof extends Mitchell's representation independence [17] using a logical relation with a special care on monads. Let $\mathcal{A} = (\mathbb{C}, \mathcal{T}, A, k, \alpha)$ be a $\lambda_c(B, \Sigma)$-structure and (V, C) be a property over $\mathcal{A}$ that satisfies (I), (C1) and (C2). We aim to construct a $\lambda_c(B, \Sigma)$-structure $\mathcal{D} = (\mathbb{E}, \dot{\mathcal{T}}, V, \dot{k}, \dot{\alpha})$ such that 1) $\dot{T}\dot{f}, \dot{\eta}_X, \dot{\mu}_X, \dot{\theta}_{X,Y}$ are respectively above $T(p\dot{f}), \eta_{pX}, \mu_{pX}, \theta_{pX,pY}$, 2) $\dot{k}c$ is above kc, 3) $\dot{\alpha}o_X$ is above αo_{pX} and 4) $\dot{T}Vb \leq Cb$ holds in $\mathbb{E}_{TAb}$.

We construct the strong monad $\dot{\mathcal{T}}$ on $\mathbb{E}$ by *categorical* $\top\top$*-lifting* [11], which is a semantic formulation of Lindley and Stark's $\top\top$-lifting [15,14]. Let $X \in \mathbb{E}$ be above $I \in \mathbb{C}$. We first define the object $X^{\top\top(Cb)}$ above TI to be the inverse image of $(X \dot{\Rightarrow} Cb) \dot{\Rightarrow} Cb$ along the following morphism $\sigma_I^{\mathcal{T},Ab} : TI \to (I \Rightarrow TAb) \Rightarrow TAb$:

$$\sigma_I^{\mathcal{T},Ab} = \lambda(ev^{\#} \circ \theta_{I \Rightarrow TAb, I} \circ \langle \pi_2, \pi_1 \rangle).$$

This is the strong monad morphism (see Section 4) from $\mathcal{T}$ to the continuation monad $(- \Rightarrow TAb) \Rightarrow TAb$. We then define $\dot{T}X$ by the following fibrewise product:

$$\dot{T}X = \bigwedge_{b \in B} X^{\top\top(Cb)} \left(= \bigwedge_{b \in B} (\sigma_I^{\mathcal{T},Ab})^*((X \dot{\Rightarrow} Cb) \dot{\Rightarrow} Cb) \right). \tag{1}$$

Proposition 2 ([11]). *Let $X, Y \in \mathbb{E}$.*

1. *For any morphism f in $\mathbb{C}$, $f : X \dot{\to} Y$ implies $Tf : \dot{T}X \dot{\to} \dot{T}Y$.*
2. *We have $\eta_{pX} : X \dot{\to} \dot{T}X$, $\mu_{pX} : \dot{T}\dot{T}X \dot{\to} \dot{T}X$ and $\theta_{pX,pY} : X \dot{\times} \dot{T}Y \dot{\to} \dot{T}(X \dot{\times} Y)$.*

From this, for any morphism $\dot{f} : X \to Y$ in $\mathbb{E}$, we define $\dot{T}\dot{f}$ to be the witness of $T(p\dot{f}) : \dot{T}X \dot{\to} \dot{T}Y$. We similarly define morphisms $\dot{\eta}_X, \dot{\mu}_X, \dot{\theta}_{X,Y}$ in $\mathbb{E}$ to be the witnesses of the statements in Proposition 2-2. Then the tuple $\dot{\mathcal{T}} = (\dot{T}, \dot{\eta}, \dot{\mu}, \dot{\theta})$ forms a strong monad over $\mathbb{E}$ satisfying the condition 1.

From (C1), for each $(\boldsymbol{b}, a) \in B^* \times (B^*)^*$ and $c \in \Sigma^{\boldsymbol{b} \to a}$, we define $\dot{k}c$ to be the witness of $kc : \prod_{i=1}^{|\boldsymbol{b}|} Vb_i \dot{\to} Va$. This defines the component $\dot{k}$ of $\mathcal{D}$ satisfying the condition 2.

We construct the component $\dot{\alpha}$ of $\mathcal{D}$ satisfying the condition 3. We first prove a general fact about algebraic operations for the $\top\top$-lifted monads $\dot{T}$:

Proposition 3. *Let $Z \in \mathbb{C}$, α be an Z-ary algebraic operation for $\mathcal{T}$ and $\dot{Z} \in \mathbb{E}$ be above Z. If $\alpha_{Ab} : \dot{Z} \dot{\Rightarrow} Cb \dot{\to} Cb$ holds for all base types $b \in B$, then for any object $X \in \mathbb{E}$, we have $\alpha_{pX} : \dot{Z} \dot{\Rightarrow} \dot{T}X \dot{\to} \dot{T}X$.* [3]

For any $a \in (B^*)^*$, Va is above Aa. Therefore from (C2) and Proposition 3, for any arity $a \in (B^*)^*$, algebraic operation symbol $o \in \Sigma^a$ and $X \in \mathbb{E}$, we have $\alpha o_{pX} : Va \dot{\Rightarrow} \dot{T}X \dot{\to} \dot{T}X$. We define $\dot{\alpha}o_X$ to be its witness. Then $\dot{\alpha}o$ is a Va-ary algebraic operation for $\dot{\mathcal{T}}$.

We have obtained the $\lambda_c(B, \Sigma)$-structure $\mathcal{D}$ satisfying the conditions 1–3. One can easily show the basic lemma of logical relations:

Proposition 4. *For any well-typed $\lambda_c(B, \Sigma)$-term $x_1 : \rho_1, \cdots, x_n : \rho_n \vdash M : \rho$, we have $\mathcal{A}[\![M]\!] : \mathcal{D}[\![\rho_1]\!] \dot{\times} \cdots \dot{\times} \mathcal{D}[\![\rho_n]\!] \dot{\to} \dot{T}\mathcal{D}[\![\rho]\!]$; its witness is given by $\mathcal{D}[\![M]\!]$.*

We finally show the condition 4.

Proposition 5. *For any base type $b \in B$, we have $\dot{T}Vb \leq Cb$ in $\mathbb{E}_{TAb}$.*

Proof. As $\dot{T}Vb = \bigwedge_{b' \in B} T^{\top\top(Cb')}Vb$, it is sufficient to show $T^{\top\top(Cb)}Vb \leq Cb$. Let us write $\dot{\eta}_b : Vb \to Cb$ for the witness of $\eta_{Ab} : Vb \dot{\to} Cb$. Then in $\mathbb{E}$ we obtain a morphism

$$\dot{ev} \circ \dot{\langle} \mathrm{id}, \dot{\lambda}(\dot{\eta}_b \circ \dot{\pi}_2) \dot{\rangle} \circ \overline{\sigma_{Ab}^{\mathcal{T},Ab}}((Vb \dot{\Rightarrow} Cb) \dot{\Rightarrow} Cb) : T^{\top\top(Cb)}Vb \to Cb$$

which is above $ev \circ \langle \mathrm{id}, \lambda(\eta_{Ab} \circ \pi_2) \rangle \circ \sigma_{Ab}^{\mathcal{T},Ab} = \mathrm{id}_{TAb}$. Therefore $T^{\top\top(Cb)}Vb \leq Cb$.

Theorem 1 is an immediate corollary of Proposition 4 and 5. This ends the proof.

[3] Though we do not use it, the converse of this statement holds: if $\alpha_{pX} : \dot{Z} \dot{\Rightarrow} \dot{T}X \dot{\to} \dot{T}X$ holds for all $X \in \mathbb{E}$, then $\alpha_{Ab} : \dot{Z} \dot{\Rightarrow} Cb \dot{\to} Cb$ holds for any base type $b \in B$.

4 Effect Simulation by Monad Morphism

Monadic semantics are often related by *strong monad morphisms*. Let $\mathcal{T}_i = (T_i, \eta_i, \mu_i, \theta_i)$ be strong monads ($i = 1, 2$) over a cartesian category $\mathbb{C}$. A strong monad morphism from $\mathcal{T}_1$ to $\mathcal{T}_2$ is a natural transformation $\sigma : T_1 \to T_2$ such that

$$\sigma_I \circ (\eta_1)_I = (\eta_2)_I \quad \sigma_I \circ (\mu_1)_I = (\mu_2)_I \circ \sigma_{T_2 I} \circ T_1 \sigma_I \quad (\theta_2)_{I,J} \circ I \times \sigma_J = \sigma_{I \times J} \circ (\theta_1)_{I,J}.$$

It transfers each Z-ary algebraic operation α for $\mathcal{T}_1$ to the following Z-ary algebraic operation $\sigma\alpha$ for $\mathcal{T}_2$:

$$(\sigma\alpha)_I = \; Z \Rightarrow T_2 I \xrightarrow{Z \Rightarrow (\eta_1)_{T_2 I}} Z \Rightarrow T_1 T_2 I \xrightarrow{\alpha_{T_2 I}} T_1 T_2 I \xrightarrow{\sigma_{T_2 I}} T_2 T_2 I \xrightarrow{(\mu_2)_I} T_2 I.$$

We define the *image* of a $\lambda_c(B, \Sigma)$-structure $\mathcal{A}_1 = (\mathbb{C}, \mathcal{T}_1, A, k, \alpha)$ along σ to be the $\lambda_c(B, \Sigma)$-structure $\sigma\mathcal{A}_1 = (\mathbb{C}, \mathcal{T}_2, A, k, \sigma\alpha)$, where $\sigma\alpha$ assigns the algebraic operation $\sigma(\alpha o)$ to each algebraic operation symbol $o \in \Sigma^a$ of arity $a \in (B^*)^*$.

Theorem 2. *Let $\mathcal{A} = (\mathbb{C}, \mathcal{T}_1, A, k, \alpha)$ be a $\lambda_c(B, \Sigma)$-structure such that $\mathbb{C}$ is small, $\mathcal{T}_2$ be a strong monad over $\mathbb{C}$ and $\sigma : \mathcal{T}_1 \to \mathcal{T}_2$ be a strong monad morphism. Then for any well-typed term $x_1 : b_1, \cdots, x_n : b_n \vdash M : b$, we have $\sigma \circ \mathcal{A}[\![M]\!] = (\sigma\mathcal{A})[\![M]\!]$.*

Proof. We pull-back the subobject fibration $\mathbf{Sub}([\mathbb{C}^{op}, \mathbf{Set}]) \to [\mathbb{C}^{op}, \mathbf{Set}]$ along the finite-product preserving functor $D : \mathbb{C}^2 \to [\mathbb{C}^{op}, \mathbf{Set}]$ defined by $D(I, J) = yI \times yJ$; here y is yoneda embedding. From Proposition 1, the leg of the pullback, say $q : \mathbb{K} \to \mathbb{C}^2$, is a category for logical relations. Now the following simulation (V, C) between $\mathcal{A}$ and $\sigma\mathcal{A}$ satisfies (I), (C1) and (C2):

$$VbH = \{(f, f) \mid f \in \mathbb{C}(H, Ab)\} \quad CbH = \{(f, \sigma_{Ab} \circ f) \mid f \in \mathbb{C}(H, T_1 Ab)\} \qquad (H \in \mathbb{C})$$

The goal is a corollary of Theorem 1 with the above simulation.

Example 4. (Continued from Example 1) There is a monad morphism σ from $\mathcal{T}_m$ to $\mathcal{T}_p$ mapping a list $l \in T_m I$ to the set $\sigma_I(l) \in T_p I$ of elements occurring in l. Thus from Theorem 2, for any well-typed term $x_1 : b_1, \cdots, x_n : b_n \vdash M : b$ and value $v_i \in Ab_i$, $\mathcal{A}_1[\![M]\!](\boldsymbol{v})$ is the set of elements occurring in $\mathcal{A}_2[\![M]\!](\boldsymbol{v})$.

Example 5. Let $\mathcal{A} = (\mathbb{C}, \mathcal{T}, A, k, \alpha)$ be a $\lambda_c(B, \Sigma)$-structure such that $\mathbb{C}$ is small. We write $C^{\mathcal{T}, \perp}$ for the *continuation monad* with respect to the monad $\mathcal{T}$ and a result type $\perp$ (which is just an object in $\mathbb{C}$). The functor part of $C^{\mathcal{T}, \perp}$ is given by $C^{\mathcal{T}, \perp} I = (I \Rightarrow T\perp) \Rightarrow T\perp$. As we have seen in the proof of Theorem 1, there is a strong monad morphism $\sigma^{\mathcal{T}, \perp} : \mathcal{T} \to C^{\mathcal{T}, \perp}$. We instantiate Theorem 2 with it, and obtain an equation $\sigma^{\mathcal{T}, \perp} \circ \mathcal{A}[\![M]\!] = \sigma^{\mathcal{T}, \perp}\mathcal{A}[\![M]\!]$, particularly for any closed M. The r.h.s. of this equation is the *CPS semantics* [2] of $\lambda_c(B, \Sigma)$, while the l.h.s. roughly corresponds to $\lambda k \,.\, k^{\#}(\mathcal{A}[\![M]\!])$. This equation is indeed the *monadic congruence result* [2] restricted to base types.

5 Comparing Two Monadic Semantics of ν-Calculus

Dynamic name creation, such as the one in π-calculus, is often categorically modelled in the presheaf category over the category $\mathcal{I}$ of finite sets and injections between them

Table 1. Definition of two ν-calculus structures

	$\mathcal{A}_1$	$\mathcal{A}_2$
Category	$[\mathcal{I}, \mathbf{Set}]$	$\mathbf{Set}$
Monad	$T_1 F = \mathrm{colim}_{Q \in \mathcal{I}} F(- + Q)$	$T_2 I = \mathbf{N} \Rightarrow I \times \mathbf{N}$
Name type object	$A_1\mathbf{n} = N : \mathcal{I} \hookrightarrow \mathbf{Set}$	$A_2\mathbf{n} = \mathbf{N}$
Name equality predicate	$(k_1\mathbf{eq})_P(i,j) = \begin{cases} \iota_1(*) & (i = j) \\ \iota_2(*) & (i \neq j) \end{cases}$	$(k_2\mathbf{eq})(i,j) = \begin{cases} \iota_1(*) & (i = j) \\ \iota_2(*) & (i \neq j) \end{cases}$
Name creation	$\alpha_1\nu$: see (2) below	$\alpha_2\nu = \lambda f x \,.\, f(x)(x+1)$

[25,26]. On the other hand, in practical programming names are represented by natural numbers and dynamic name creation is implemented by a hidden global counter that keeps track of the next fresh name.

In this section, we consider Stark's *ν-calculus* [25] and discuss an effect simulation problem between presheaf semantics and global counter semantics of name creation. The ν-calculus has only one base type $\mathbf{n}$ for *names*, one effect-free constant $\mathbf{eq} : \mathbf{n} \times \mathbf{n} \to 1 + 1$ for checking name equality, and one algebraic operation $\nu(x^{\mathbf{n}}.M)$ whose intended meaning is to allocate a fresh name and bind it to x, like the one in π-calculus. We write Σ_ν for the signature specifying only these symbols. The ν-calculus is then defined to be $\lambda_c(\{\mathbf{n}\}, \Sigma_\nu)$. Below we call a $\lambda_c(\{\mathbf{n}\}, \Sigma_\nu)$-structure *$\nu$-calculus structure*.

In Table 1 we present two ν-calculus structures with which we consider an effect simulation problem. The ν-calculus structure $\mathcal{A}_1$ extracts the ingredients that are used in the categorical semantics of the ν-calculus in [25]. The monad $\mathcal{T}_1$ is Stark's *dynamic name creation monad*:

$$T_1 FP = \mathrm{colim}_{Q \in \mathcal{I}} F(P + Q) = \{(Q, x) \mid Q \in \mathcal{I}, x \in F(P + Q)\}/\sim$$

where $(Q, x) \sim (R, y)$ if there are $S \in \mathcal{I}$ and two injections $l : Q \rightarrowtail S, m : R \rightarrowtail S$ such that $F(P + l)(x) = F(P + m)(y)$. We note $TNP \simeq N(P + 1)$. The object for the name type is the inclusion functor $N : \mathcal{I} \hookrightarrow \mathbf{Set}$; this is the standard choice for representing names. The behaviour of the name equality predicate at a finite set P is given in Table 1; there i, j are elements in P. The algebraic operation $\alpha_1\nu$ for name creation is defined by

$$((\alpha_1\nu)_F)_P(\alpha) = [1 + Q, F(i)(y)]_\sim \quad (\text{where } [Q, y]_\sim = \alpha_{P+1}(\iota_1, \iota_2)) \tag{2}$$

where $i : (P + 1) + Q \to P + (1 + Q)$ is the coherence isomorphism.

The ν-calculus structure $\mathcal{A}_2$ is a semantic analogue of dynamic name creation by a global state. We note that the interpretation $\mathcal{A}_2[\![-]\!]$ is not sound with respect to the ν-calculus axioms in [25].

We compare the denotation of a well-typed term $x_1 : \mathbf{n}, \ldots, x_n : \mathbf{n} \vdash M : \mathbf{n}$ in each ν-calculus structure. Suppose that p names have been allocated, and some of them are supplied to the free variables of M. Then M returns either one of the allocated names supplied to its free variables, or M allocates a new name and returns it. This behaviour is expressed differently in each ν-calculus structure:

- (in $\mathcal{A}_1$) Let P be the finite set consisting of p allocated names. We feed $i_1, \ldots, i_n \in P$ to the free variables of M. When M returns an allocated name, the denotation $\mathcal{A}_1[\![M]\!]_P(\boldsymbol{i}) \in TNP \simeq P + 1$ is $\iota_1(i)$ with some $i \in P$. Otherwise, M returns a new name and the denotation is $\iota_2(*)$.
- (in $\mathcal{A}_2$) Natural numbers $0, \ldots, p-1$ correspond to the allocated names. We thus feed $0 \leq i_1 \ldots i_n < p$ to the free variables of M. The global counter pointing to the next fresh name is now p, so the name that M returns is given by $i = \pi_1(\mathcal{A}_2[\![M]\!](\boldsymbol{i})(p))$. When M returns an allocated name, $0 \leq i < p$; otherwise $i \geqslant p$. In fact, this behaviour of M remains the same even when the counter is increased from p. Therefore when M returns an allocated name i, for any $k \geq p$ we have $\pi_1(\mathcal{A}_2[\![M]\!](\boldsymbol{i})(k)) = i$; otherwise for any $k \geq p$ we have $\pi_1(\mathcal{A}_2[\![M]\!](\boldsymbol{i})(k)) \geq k$.

Based on this analysis, we establish a correspondence between the denotation of M in each ν-calculus structure. As names are represented differently, this relationship is parametrised by bijective correspondences between allocated names and natural numbers. Below for a finite set P, by $|P|$ we mean its cardinality. For a natural number p, we write $\overline{p}$ for the finite set $\{0, \cdots, p-1\}$. A *name enumeration* is a bijection $\sigma : P \to \overline{|P|}$.

Theorem 3. *Let $x_1 : \mathbf{n}, \ldots, x_n : \mathbf{n} \vdash M : \mathbf{n}$ be a ν-calculus term. For any finite set P, elements $i_1 \ldots i_n \in P$ and a name enumeration $\sigma : P \to \overline{|P|}$, either*

- *there is $i \in P$ such that $\mathcal{A}_1[\![M]\!]_P(\boldsymbol{i}) = \iota_1(i)$ and for all $k \geq |P|$, we have $\pi_1 \circ \mathcal{A}_2[\![M]\!](\sigma(\boldsymbol{i}))(k) = \sigma(i)$, or*
- *$\mathcal{A}_1[\![M]\!]_P(\boldsymbol{i}) = \iota_2(*)$ and for all $k \geq |P|$, we have $\pi_1 \circ \mathcal{A}_2[\![M]\!](\sigma(\boldsymbol{i}))(k) \geqslant k$.*

The rest of this section is the proof of this theorem. We construct a suitable category for logical relations over $[\mathcal{I}, \mathbf{Set}] \times \mathbf{Set}$, and give a simulation (V, C) between $\mathcal{A}_1$ and $\mathcal{A}_2$ that implies the goal of the theorem. We then check that it satisfies (I), (C1) and (C2).

We observe that the theorem is parametrised by name enumerations, so we first introduce the category $\mathcal{E}$ of name enumerations. As defined above, a name enumeration is a bijection $\sigma : P \to \overline{|P|}$. A morphism h from σ to $\tau : Q \to \overline{|Q|}$ is a (necessarily unique) injection $h : P \to Q$ such that $\sigma = \tau \circ h$. We note that $\mathcal{E}$ is actually equivalent to $(\mathbf{N}, \leq)$. There is an evident projection functor $\pi : \mathcal{E} \to \mathcal{I}$.

We next pull-back the subobject fibration $\mathbf{Sub}([\mathcal{E}, \mathbf{Set}]) \to [\mathcal{E}, \mathbf{Set}]$ along the finite-product preserving functor $D : [\mathcal{I}, \mathbf{Set}] \times \mathbf{Set} \to [\mathcal{E}, \mathbf{Set}]$ defined by $D(F, I) = (F \circ \pi) \times \Delta I$. We obtain the category $q : \mathbf{ERel} \to [\mathcal{I}, \mathbf{Set}] \times \mathbf{Set}$ for logical relations by Proposition 1. An object in $\mathbf{ERel}$ is a triple (X, F, I) where $F \in [\mathcal{I}, \mathbf{Set}]$, $I \in \mathbf{Set}$ and X assigns a binary relation $X\sigma \subseteq FP \times I$ to each name enumeration $\sigma : P \to \overline{|P|}$. Moreover, X should satisfy the *monotonicity condition*: for any $h \in \mathcal{E}(\sigma, \tau)$ and $(x, y) \in X\sigma$, we have $(Fhx, y) \in X\tau$.

We give the simulation (V, C) between $\mathcal{A}_1$ and $\mathcal{A}_2$ that entails Theorem 3. For each name enumeration $\sigma : P \to \overline{|P|}$, we define $V\mathbf{n}\sigma = \{(i, \sigma(i)) \mid i \in P\}$ and

$$C\mathbf{n}\sigma = \{(\iota_1(i), f) \mid i \in P \wedge \forall k \geq |P| \,.\, \pi_1 \circ f(k) = \sigma(i)\} \cup \{(\iota_2(*), f) \mid \forall k \geq |P| \,.\, \pi_1 \circ f(k) \geq k\}.$$

Proposition 6. *The above simulation (V, C) satisfies (I), (C1) and (C2).*

Theorem 3 is an immediate corollary of Theorem 1 with the above simulation.

$$\frac{\Gamma, f : \rho \to \sigma, x : \rho \vdash M : \sigma}{\Gamma \vdash \mu f x.M : \rho \to \sigma} \qquad \begin{array}{l} \mathcal{A}[\![\Gamma \vdash \mu f x.M : \rho \Rightarrow \sigma]\!] \\ = \mathbf{fix}^{\mathcal{A}[\![\Gamma]\!]}_{\mathcal{A}[\![\rho \Rightarrow \sigma]\!]}(\mathcal{A}[\![\lambda x\,.\,M]\!]^{\#} \circ \theta_{\mathcal{A}[\![\Gamma]\!], \mathcal{A}[\![\rho \Rightarrow \sigma]\!]}). \end{array}$$

Fig. 1. Recursion: Typing Rule and Interpretation

6 Extending $\lambda_c(B, \Sigma)$ with Recursive Functions

We next add the recursive function constructor $\mu f x.M$ to $\lambda_c(B, \Sigma)$. This term creates a closure that may recursively call itself inside M; see Figure 1 for its typing rule. We call the extended calculus $\lambda_c^{\mathbf{fix}}(B, \Sigma)$. To interpret the recursion under the presence of computation, we employ *uniform T-fixpoint operator* [24]. An equivalent, direct formulation of recursion in call-by-value is also studied in [8]. Let $\mathcal{T} = (T, \eta, \mu, \theta)$ be a strong monad over a cartesian category $\mathbb{C}$. A *uniform T-fixpoint operator* for $\mathcal{T}$ is a family of mappings $\mathbf{fix}_I : \mathbb{C}(TI, TI) \to \mathbb{C}(1, TI)$ such that $\mathbf{fix}_I(f) = f \circ \mathbf{fix}_I(f)$, and it satisfies the *uniformity principle*: for any $f : TI \to TI, g : TJ \to TJ$ and $h : I \to TJ$, $h^{\#} \circ f = g \circ h^{\#}$ implies $g = h^{\#} \circ \mathbf{fix}_I(f)$. When $\mathbb{C}$ is a CCC, we can parametrise it as $\mathbf{fix}_I^X : \mathbb{C}(X \times TI, TI) \to \mathbb{C}(X, TI)$; see [24] for the detail.

Definition 5. *A $\lambda_c^{\mathbf{fix}}(B, \Sigma)$-structure is a pair of a $\lambda_c(B, \Sigma)$-structure $\mathcal{A}$ and a uniform T-fixpoint operator* **fix** *for the strong monad of $\mathcal{A}$.*

The interpretation of a recursive function constructor is given in Figure 1.

We aim to extend Theorem 1 to $\lambda_c^{\mathbf{fix}}(B, \Sigma)$ and a $\lambda_c^{\mathbf{fix}}(B, \Sigma)$ structure $\mathcal{A}$ where 1) the category of $\mathcal{A}$ is ω**CPO**-*enriched*[4], 2) the strong monad of $\mathcal{A}$ *lifts* the given domain and 3) the uniform T-fixpoint operator is given by the least fixpoint. That $\mathcal{T}$ lifts a given domain (as defined below) is expressed by the fact that $\mathcal{T}$ admits an algebraic operation denoting the least element of TI.

Definition 6. *We call a strong monad $\mathcal{T}$ over a* **Pos**-*enriched bi-CCC $\mathbb{C}$* pseudo-lifting *if it has an* 0-*ary algebraic operation $\bot$ such that for any $I \in \mathbb{C}$, $\bot_I$ is the least element in $\mathbb{C}(0 \Rightarrow TI, TI) \simeq \mathbb{C}(1, TI)$.*

We note that for any pseudo-lifting monad $(\mathcal{T}_1, \bot)$ over a **Pos**-enriched bi-CCC $\mathbb{C}$, $\mathcal{T}_2$ be another strong monad over $\mathbb{C}$ and strong monad morphism $\sigma : \mathcal{T}_1 \to \mathcal{T}_2$, the pair $(\mathcal{T}_2, \sigma\bot)$ is pseudo-lifting and σ is strict, that is, $\sigma_I \circ \bot_I = (\sigma\bot)_I$.

Definition 7. *An ω**CPO**-enriched $\lambda_c(B, \Sigma)$-structure is a tuple $(\mathbb{C}, \mathcal{T}, A, k, \alpha, \bot)$ such that the first five components form a $\lambda_c(B, \Sigma)$-structure, $\mathbb{C}$ is an ω**CPO**-enriched bi-CCC and $(\mathcal{T}, \bot)$ is a pseudo-lifting monad.*

We can turn every ω**CPO**-enriched $\lambda_c(B, \Sigma)$-structure into a $\lambda_c^{\mathbf{fix}}(B, \Sigma)$-structure by paring it with the uniform T-fixpoint operator given by $\mathbf{fix}_I(f) = \bigsqcup_{n \in \mathbf{N}} f^{(n)} \circ \bot_I$.

Let $\mathcal{A} = (\mathbb{C}, \mathcal{T}, A, k, \alpha, \bot)$ be an ω**CPO**-enriched $\lambda_c(B, \Sigma)$-structure, and $p : \mathbb{E} \to \mathbb{C}$ be a category for logical relations. Since p is faithful, we can restrict the partial order

[4] We write **Pos** for the category of posets and monotone functions with finite products as the symmetric monoidal structure. The category ω**CPO** is a subcategory of **Pos** where objects are ω-complete partial orders and and morphisms are continuous functions.

on $\mathbb{C}(pX, pY)$ to $\mathbb{E}(X, Y)$. Moreover, as p strictly preserves bi-cartesian closed structure, $\mathbb{E}$ becomes a **Pos**-enriched bi-CCC. We call $X \in \mathbb{E}$ above TI *admissible* if 1) $\bot_I : \dot{1} \dot{\to} X$ and 2) for any $Y \in \mathbb{E}$ and ω-chain $\dot{f_i}$ in $\mathbb{E}(Y, X)$, we have $\bigsqcup_{i=0}^{\infty}(p\dot{f_i}) : Y \to X$.

Theorem 4. *Let $\mathcal{A} = (\mathbb{C}, \mathcal{T}, A, k, \alpha, \bot)$ be an ω**CPO**-enriched $\lambda_c(B, \Sigma)$-structure, $p : \mathbb{E} \to \mathbb{C}$ be a category for logical relations, (V, C) be a property over $(\mathbb{C}, \mathcal{T}, A, k, \alpha)$ such that it satisfies (I), (C1) and (C2) and Cb is admissible for all base types $b \in B$. Then for any well-typed $\lambda_c^{\mathbf{fix}}(B, \Sigma)$-term $x_1 : b_1, \cdots, x_n : b_n \vdash M : b$, we have $\mathcal{A}[\![M]\!] : Vb_1 \dot{\times} \cdots \dot{\times} Vb_n \dot{\to} Ob$.*

Theorem 5. *Let $\mathcal{A} = (\mathbb{C}, \mathcal{T}_1, k, A, \alpha, \bot)$ be an ω**CPO**-enriched $\lambda_c(B, \Sigma)$-structure such that $\mathbb{C}$ is small, $\mathcal{T}_2$ be a strong monad over $\mathbb{C}$ and $\sigma : \mathcal{T}_1 \to \mathcal{T}_2$ be a strong monad morphism. Then for any well-typed $\lambda_c^{\mathbf{fix}}(B, \Sigma)$-term $x_1 : b_1, \cdots, x_n : b_n \vdash M : b$, we have $\sigma \circ \mathcal{A}[\![M]\!] = (\sigma\mathcal{A})[\![M]\!]$.*

7 Related Work

Filinski is one of the pioneers in logical relations for monads [3], and developed various techniques to establish relationships between semantics of higher-order languages with effects [2,3,4,5,6]. He pointed out at least three methods to obtain logical relations for monads: 1) When T is a syntactically constructed monad, such as state monad and continuation monad, then define a logical relation $\dot{T}$ for T in the same way as T is constructed. 2) For a logical relation $\dot{T}$ for a monad T and a strong monad morphism $\sigma : S \to T$, the inverse image $\sigma^*\dot{T}$ is a logical relation for S. 3) For a family of logical relations $\dot{T}_i$ for a monad T, the intersection $\bigwedge_i \dot{T}_i$ is again a logical relation for T. A method based on factorisation systems is also proposed by Larrecq et al [13].

The categorical $\top\top$-lifting is technically a particular combination of the methods 1–3 (in fibrational category theory). However, what is new about $\top\top$-lifting is that we *use* the simulation / property we would like to establish on computational effects to define the logical relation $\dot{T}$; see definition (1). This idea is a secret recipe in the proofs of various results by the precursors of categorical $\top\top$-lifting, such as biorthogonality [7,16,20], $\top\top$-closure [21] and leapfrog method [14,15]; see also [12].

The advantage of logical relations for monads by $\top\top$-lifting is that it does not limit the form of simulation / property we would like to establish on computational effects. Furthermore, Proposition 3 gives a good characterisation of when algebraic operations are related by the logical relations given by $\top\top$-lifting. On the other hand, it is rather difficult to check whether non-algebraic operations that manipulate computational effects, such as Felleisen's $\mathcal{C}$- and $\mathcal{A}$-operators, are related by the logical relations given by $\top\top$-lifting; this shall be discussed in a separate paper. Extending our results with recursive types and handlers for algebraic effects [23] is also a future work.

Acknowledgement. The author is grateful to Masahito Hasegawa, Naohiko Hoshino and Susumu Nishimura for fruitful discussions. He thanks anonymous reviewers for valuable comments. This research was partly supported by Grant-in-Aid for Young Scientists (B) 20700012.

References

1. Benton, N., Hughes, J., Moggi, E.: Monads and effects. In: Barthe, G., Dybjer, P., Pinto, L., Saraiva, J. (eds.) APPSEM 2000. LNCS, vol. 2395, pp. 42–122. Springer, Heidelberg (2002)
2. Filinski, A.: Representing monads. In: Proc. POPL 1994, pp. 446–457 (1994)
3. Filinski, A.: Controlling Effects. PhD thesis, Carnegie Mellon University (1996)
4. Filinski, A.: Representing layered monads. In: Proc. POPL 1999, pp. 175–188 (1999)
5. Filinski, A.: On the relations between monadic semantics. Theor. Comput. Sci. 375(1-3), 41–75 (2007)
6. Filinski, A., Støvring, K.: Inductive reasoning about effectful data types. In: Hinze, R., Ramsey, N. (eds.) ICFP, pp. 97–110. ACM, New York (2007)
7. Girard, J.-Y.: Linear logic. Theor. Comp. Sci. 50, 1–102 (1987)
8. Hasegawa, M., Kakutani, Y.: Axioms for recursion in call-by-value. Higher-Order and Symbolic Computation 15(2-3), 235–264 (2002)
9. Hermida, C.: Fibrations, Logical Predicates and Indeterminants. PhD thesis, University of Edinburgh (1993)
10. Jacobs, B.: Categorical Logic and Type Theory. Elsevier, Amsterdam (1999)
11. Katsumata, S.: A semantic formulation of TT-lifting and logical predicates for computational metalanguage. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 87–102. Springer, Heidelberg (2005)
12. Katsumata, S.: A characterisation of lambda definability with sums via TT-closure operators. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 278–292. Springer, Heidelberg (2008)
13. Larrecq, J.-G., Lasota, S., Nowak, D.: Logical relations for monadic types. Math. Struct. in Comp. Science 18, 1169–1217 (2008)
14. Lindley, S.: Normalisation by Evaluation in the Compilation of Typed Functional Programming Languages. PhD thesis, University of Edinburgh (2004)
15. Lindley, S., Stark, I.: Reducibility and TT-lifting for computation types. In: Urzyczyn, P. (ed.) TLCA 2005. LNCS, vol. 3461, pp. 262–277. Springer, Heidelberg (2005)
16. Melliès, P.-A., Vouillon, J.: Recursive polymorphic types and parametricity in an operational framework. In: LICS, pp. 82–91. IEEE Computer Society, Los Alamitos (2005)
17. Mitchell, J.: Representation independence and data abstraction. In: Proc. POPL 1986, pp. 263–276 (1986)
18. Mitchell, J.: Foundations for Programming Languages. MIT Press, Cambridge (1996)
19. Mitchell, J., Scedrov, A.: Notes on sconing and relators. In: Martini, S., Börger, E., Kleine Büning, H., Jäger, G., Richter, M.M. (eds.) CSL 1992. LNCS, vol. 702, pp. 352–378. Springer, Heidelberg (1993)
20. Parigot, M.: Proofs of strong normalisation for second order classical natural deduction. Journal of Symbolic Logic 62(4), 1461–1479 (1997)
21. Pitts, A.: Parametric polymorphism and operational equivalence. Mathematical Structures in Computer Science 10(3), 321–359 (2000)
22. Plotkin, G., Power, J.: Semantics for algebraic operations. Electr. Notes Theor. Comput. Sci. 45 (2001)
23. Plotkin, G.D., Pretnar, M.: Handlers of algebraic effects. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 80–94. Springer, Heidelberg (2009)
24. Simpson, A., Plotkin, G.: Complete axioms for categorical fixed-point operators. In: LICS, pp. 30–41 (2000)
25. Stark, I.: Categorical models for local names. Lisp and Symbolic Computation 9(1), 77–107 (1996)
26. Stark, I.: A fully abstract domain model for the π-calculus. In: Proc. LICS 1996, pp. 36–42. IEEE, Los Alamitos (1996)

Constructing Differential Categories and Deconstructing Categories of Games

Jim Laird[1], Giulio Manzonetto[2], and Guy McCusker[1]

[1] Department of Computer Science, University of Bath, Bath, BA2 7AY, UK
[2] Radboud University, Intelligent Systems, Nijmegen, The Netherlands

Abstract. We present an abstract construction for building differential categories useful to model resource sensitive calculi, and we apply it to categories of games. In one instance, we recover a category previously used to give a fully abstract model of a nondeterministic imperative language. The construction exposes the differential structure already present in this model. A second instance corresponds to a new Cartesian differential category of games. We give a model of a Resource PCF in this category and show that it enjoys the finite definability property. Comparison with a relational semantics reveals that the latter also possesses this property and is fully abstract.

1 Introduction

An important aim in studying higher-order computation is to understand and control the way resources are used. One way to do this is by studying calculi designed to capture resource usage, and their denotational models. Two such calculi — the *differential λ-calculus* [6] of Ehrhard and Regnier and the *resource calculus* introduced by Tranquilli [13] are fundamentally related at the semantic level [5]: both may be interpreted using the notion of *differential category* introduced by Blute, Cockett and Seely [3]. In this paper, we study these concepts on both abstract and concrete levels. We give a construction of a differential category from any symmetric monoidal category, and use it to investigate the structure of newly discovered differential categories, relate them to existing examples, and to prove full abstraction results for *Resource PCF*, a typed programming language based on the resource calculus.

A potential source of differential categories, although not investigated hitherto, is *game semantics*: resource usage is represented rather explicitly in games and strategies. Indeed, we show that an existing games model of Idealized Algol with non-determinism, introduced by Harmer and McCusker [8] contains a differential Cartesian operator [4], and may therefore be used to interpret Resource PCF, although this interpretation contains non-definable finitary elements.

We then present the construction which we shall use to analyze differential categories. Its key step takes a symmetric monoidal category with countable biproducts, embeds it in its *Karoubi envelope* (idempotent splitting) and then constructs the *cofree cocommutative comonoid* on this category and a differential operator on the Kleisli category of the corresponding comonad. Since

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 186–197, 2011.

biproducts may be added to any category by free constructions, we have a way of embedding any symmetric monoidal (closed) category in a Cartesian (closed) differential category.

Although this construction is somewhat elaborate, it provides a useful tool for analyzing and relating more directly presented models. For example, applying it to the terminal (one object, one morphism) SMCC yields the key example of a differential category (and model of resource calculus [5]) based on the finite-multiset comonad on the category of sets and relations. We also show that our differential category of games embeds in one constructed from a simple symmetric monoidal category of games. By refining the strategies in these games to eliminate *history sensitive* behaviour, we obtain a constraint on strategies ($\sim$-closure) in our directly presented model of Resource PCF which corresponds to finite definability. Another useful observation is that any functor of symmetric monoidal categories lifts to one between the differential categories constructed from them. In particular, from the terminal functor we derive a functor from our category of games and $\sim$-closed strategies into the relational model which is shown to be *full*. From this we may deduce that the relational model of Resource PCF is *fully abstract*.

2 Differential Categories and Resource PCF

Differential categories were introduced by Blute, Cockett and Seely to formalize derivatives categorically. The authors started from monoidal categories [3], then extended the notion to Cartesian ones [4]; a further generalization to Cartesian closed categories has been made in [5] to model differential and resource λ-calculi.

Throughout this paper we will be working with categories whose hom-sets are endowed with the structure of a commutative monoid $(+, 0)$. We elide all associativity and unit isomorphisms associated with monoidal categories.

Let $\mathbf{C}$ be a commutative-monoid-enriched symmetric monoidal category, i.e. it is a symmetric monoidal category such that composition and tensor preserve the commutative monoid structure on hom-sets, so that $(f+g);h = f;h+g;h$, $k;(f+g) = k;f+k;g$, $f;0 = 0 = 0;f$, $(f+g)\otimes h = f\otimes h+g\otimes h$ and $f\otimes 0 = 0$.

A *coalgebra modality* on $\mathbf{C}$ is a comonad $(!, \delta, \epsilon)$ such that each object $!A$ is equipped with a comonoid structure $\Delta_A : !A \longrightarrow !A \otimes !A$, $e_A : !A \longrightarrow I$. In addition to the associativity and unit equations for the comonoid, it should be the case that δ is a morphism of comonoids, that is, $\delta_A; e_{!A} = e_A$ and $\delta_A; \Delta_{!A} = \Delta_A; \delta_A \otimes \delta_A$.

Given such a structure, a *differential combinator* is a family of maps $D_{A,B} : \mathbf{C}(!A, B) \to \mathbf{C}(A \otimes !A, B)$, natural in A and B and respecting the commutative monoid structure of the hom-sets, satisfying the following four axioms.

- $D(e_A) = 0$,
- $D(\Delta; f\otimes g) = (A\otimes\Delta);(D(f)\otimes g)+(A\otimes\Delta);\cong;(f\otimes D(g))$ where $f : !A \to B$, $g : !A \to C$ and $\cong$ is the appropriate symmetry map,
- $D(\epsilon_A; f) = (A \otimes e_A); f$,
- $D(\delta_A; !f; g) = (A\otimes\Delta_A);(D(f)\otimes(\delta_A; !f));D(g)$ for $f : !A \to B$ and $g : !B \to C$.

Λ^r: M, N $::= x \mid \lambda x.M \mid MP \mid \mathrm{ifz}(M,M,M) \mid \mathrm{Fix}(M) \mid$ $\mid \mathrm{succ}(M) \mid \mathrm{pred}(M) \mid \mathrm{zero}$ terms

Λ^b: P $::= [L_1, \ldots, L_\ell, N_1^!, \ldots, N_n^!]$ bags

(a) Grammar of terms, resources and bags

Evaluation contexts: $E(\!|-|\!) ::= (\!|-|\!) \mid EP \mid \lambda x.E \mid \mathrm{pred}(E) \mid \mathrm{succ}(E) \mid \mathrm{ifz}(E,M,N)$

Let contexts: $F(\!|-|\!) ::= (\!|-|\!) \mid (\lambda x.F)P$

Linear head reduction:

$$E(\!|F(\lambda x.E'(\!|x|\!))(P \uplus [N])|\!) \to E(\!|F(\lambda x.E'(\!|N|\!))P|\!)$$
$$E(\!|F(\lambda x.E'(\!|x|\!))(P \uplus [N^!])|\!) \to E(\!|F(\lambda x.E'(\!|N|\!))(P \uplus [N^!])|\!)$$
$$E(\!|F(\lambda x.\underline{n})[N_1^!, \ldots, N_k^!]|\!) \to E(\!|F(\underline{n})|\!) \text{ for some } k \geq 0,$$
$$E(\!|\mathrm{ifz}(\mathrm{zero}, M, N)|\!) \to E(\!|M|\!) \qquad E(\!|\mathrm{ifz}(\mathrm{succ}(\underline{n}), M, N)|\!) \to E(\!|N|\!)$$
$$E(\!|\mathrm{pred}(\mathrm{succ}(\underline{n}))|\!) \to E(\!|\underline{n}|\!) \qquad E(\!|\mathrm{Fix}(M)|\!) \to E(\!|M[\mathrm{Fix}(M)^!]|\!)$$

(b) Operational semantics.

Fig. 1. Syntax and operational semantics of Resource PCF

A *differential category* is a commutative-monoid-enriched symmetric monoidal category with a coalgebra modality and a differential combinator. When the coalgebra modality is a linear exponential comonad, its Kleisli category is a *Cartesian differential category* whose differential combinators are denoted by $D^\times_{A,B} : \mathbf{C}(A,B) \to \mathbf{C}(A \times A, B)$. We refer to [4] for the general definition.

A *Cartesian-closed differential category* is a Cartesian differential category with closed structure, such that the operation of currying preserves the commutative monoid structure on hom-sets and for all $f : C \times A \to B$, $D^\times(\Lambda(f)) : C \times C \to (A \Rightarrow B)$ is equal to $\Lambda(\langle \pi_0 \times 0_A, \pi_1 \times \mathrm{id}_A \rangle; D^\times(f))$. The leading examples of such categories, studied in [5], are Ehrhard's category of finiteness spaces, and the category **MRel** of "multiset relations", which is the Kleisli category for the finite-multiset comonad on the category **Rel** of sets and relations.

We now describe a simply typed resource calculus which incorporates the constants of PCF, making it a prototypical resource-sensitive programming language. Resource PCF has two syntactic categories: terms, that are in functional position, and bags, that are in argument position and represent finite multisets of resources. Figure 1(a) gives the grammar generating the set Λ^r of *terms* and the set Λ^b of *bags* (whose union is denoted by $\uplus$) together with their typical metavariables. A *resource* can be linear (it must be used exactly once) or reusable (it can be used *ad libitum*) and in the latter case is decorated with a "!" superscript.

Terms of the form $\mathrm{succ}^n(\mathrm{zero})$ are denoted by $\underline{n}$.

Types are generated by $A, B ::= \mathtt{nat} \mid A \to A$. *Environments* Γ are finite lists $x_1 : A_1; \cdots; x_n : A_n$ assigning types to variables. *Typing rules* are straightforward to define, yielding a collection of typed terms-in-context $\Gamma \vdash M : A$.

The operational semantics is defined in Figure 1(b) via a *linear head reduction*. An equivalent presentation more in the style of [13] would also be possible. We say that a closed term M of ground type *converges*, written $M{\Downarrow}$, if M reduces to $\underline{k}$ for some $k \in \omega$. We denote by $C(\!|\cdot|\!)$ arbitrary contexts (e.g., in Theorem 13).

Resource PCF can be interpreted in any cpo-enriched Cartesian closed differential category having a weak natural number object. The interpretations of the constants and constructors of PCF are standard, leaving only the application, which is treated as follows, as in [5]. In every Cartesian closed differential category it is possible to define an operator $\star$ on morphisms $f : C \times A \to B$, $g : C \to A$ setting $f \star g := \langle\langle 0_C, g \circ \pi_0\rangle, \mathrm{id}\rangle; D^{\times}(f) : C \times A \to B$. Intuitively, $f \star g$ is obtained by force-feeding the 2^{nd} argument A of f with *one copy* of the result of g. The type is not modified because $f \star g$ may still depend on A.

The interpretation $[\![M]\!]_\Gamma : [\![\Gamma]\!] \to [\![A]\!]$ of $\Gamma \vdash M : A$ is defined as usual, except for the case of application where we set:

$$[\![M[\vec{L}, \vec{N}^!]]\!]_\Gamma = \langle \mathrm{id}, \textstyle\sum_{i=1}^{n} [\![N_i]\!]_\Gamma\rangle; ((\cdots(\Lambda^-([\![M]\!]_\Gamma) \star [\![L_1]\!]_\Gamma)\cdots) \star [\![L_\ell]\!]_\Gamma).$$

3 A Cartesian Closed Differential Category of Games

In this section we recall the definition of the category of games introduced in [7,8], and show that it is a Cartesian closed differential category.

An *arena* A is a finite bipartite forest over two sets of moves, M_A^P and M_A^O with edge relation $\vdash$. We say that a move is *enabled* by its parent in the forest, and that root moves are *initial*. A *QA-arena* is an arena equipped with a map labelling each move as a question (Q) or answer (A), such that every answer is the child of a question. We assume the standard notions of *justified sequence*, *views*, *P-* and *O-visibility* from the game semantics literature; see [10] for example. Given a justified sequence s, we say that an answer-move occurrence a *answers* the question occurrence q that justifies it. A justified sequence s satisfies *P-well-bracketing* if, for every prefix $s'a$ with a an answer move by P, the question that a answers is the rightmost O-question in the view $\ulcorner s' \urcorner$; call this the *pending question* at s'. A justified sequence is *complete* if every question is answered exactly once; we write $\mathsf{comp}(A)$ for the set of complete justified sequences of A.

Lemma 1. *If s is a complete justified sequence that satisfies P-visibility (resp. O-visibility), then s satisfies P-well-bracketing (resp. O-well-bracketing).*

Proof. A simple analysis of views shows that, if q is an O-question that is answered when some later O-question q' is pending, then when q' is answered, again a later O-question is pending. There can therefore be no O-question that is answered when it is not pending, because s is finite. □

A sequence is *well-opened* if it contains exactly one initial O-move. A *strategy* for an arena A is a set of complete sequences in which O plays first, satisfying P-visibility (and, by Lemma 1, P-bracketing). Given a strategy σ, $\mathsf{wv}(\sigma)$ is the set of sequences in σ that are well-opened and satisfy O-visibility.

Given arenas A and B, we write $A \uplus B$ for the arena arising as the disjoint union of A and B, and $A^\perp$ for the arena A with O and P-moves interchanged. We can define a category $\mathbf{G}$ in which objects are arenas whose roots are all O-moves, and morphisms $A \to B$ are strategies on the arena $A^\perp \uplus B$. Composition

of strategies is the usual "parallel composition plus hiding" construction, and identities are copycat strategies. As proved in [7,8], this category is monoidal closed: disjoint union of arenas gives a tensor product, and exponentials are given by the arena $A \multimap B$, which consists of the arena B with a copy of $A^{\perp}$ attached below each initial move; duplication of $A^{\perp}$ is required to maintain the forest structure. Every object of $\mathbf{G}$ can be endowed with a comonoid structure, and the subcategory of comonoid homomorphisms is a Cartesian closed category $\mathbf{G}^{\otimes}$. These maps are those whose choice of move at any stage depends only on the current thread, that is, the subsequence of moves hereditarily justified by the initial O-move currently in view; it follows that such strategies are completely determined by the well-opened plays they contain.

Erratic Idealized Algol (EIA) is an applied typed λ-calculus with an appropriate stock of constants making it a higher-order imperative programming language with local state, consisting of variables in which natural numbers can be stored. The constants include an erratic choice operator `or` which encodes nondeterministic choice. As shown in [7], this programming language can be given denotational semantics in the category $\mathbf{G}^{\otimes}$. The interpretation of the imperative programming constants is as in the standard games model of Idealized Algol from [1], and the erratic choice operator is interpreted by union of strategies.

Theorem 2 (Full abstraction [7]). *The model of EIA in $\mathbf{G}^{\otimes}$ is sound, and moreover, for any type A:*

- *if s is a complete well-opened play of $[\![A]\!]$ satisfying visibility, there is a closed term M of type A such that $\mathsf{wv}([\![M]\!]) = \{s\}$;*
- *terms $M, N : A$ are contextually equivalent if and only if $\mathsf{wv}([\![M]\!]) = \mathsf{wv}([\![N]\!])$.*

We now exhibit the differential structure that $\mathbf{G}^{\otimes}$ possesses. Let s be a complete, well-opened play in $A^{\perp} \uplus B$ which contains at least one initial A-move. Say that a complete play s' in $A^{\perp} \uplus A^{\perp} \uplus B$ is a *derivative* of s if $\Delta; \{s'\} = \{s\}$ (where Δ is the diagonal) and s' contains one initial move in the left occurrence of $A^{\perp}$. We then define $D^{\times}(\sigma)$ as the strategy whose well-opened plays are

$$\{s' \in \mathsf{comp}(A^{\perp} \uplus A^{\perp} \uplus B) \mid s' \text{ is a derivative of some well-opened } s \in \sigma\}.$$

We can verify directly that this makes $\mathbf{G}^{\otimes}$ a Cartesian closed differential category; later we will see that this follows from a general construction. Because of the definability property of the model of EIA, it is reasonable to expect that the differential operator is programmable in EIA, and indeed it is. For terms of type $A \to \mathtt{comm}$ (`comm` is the base type of *commands*) we can define the differential operator as follows (using appropriate syntactic sugar).

$$\begin{aligned}\lambda f : A \to \mathtt{comm}.\lambda a : A.\lambda a' : A.\, &\mathtt{new}\ b := \mathtt{true} \\ &\mathtt{new}\ y := f((\mathtt{if}\ b\ \mathtt{then}\ (b := \mathtt{false}; a)\ \mathtt{else}\ a')\ \mathtt{or}\ a')\ \mathtt{in} \\ &\quad \mathtt{if}\ \neg b\ \mathtt{then}\ y\ \mathtt{else}\ \mathtt{diverge}\end{aligned}$$

In any converging execution of this code, the argument a is supplied to f exactly once, though which call to f's argument receives a is chosen nondeterministically; all other calls to f's argument receive a'.

4 Constructing Differential Categories

We now describe a construction of models of intuitionistic linear logic that are also differential categories. The main ingredient is the construction of a category which possesses a comonad delivering cofree cocommutative comonoids.

Let $\mathbf{C}$ be a symmetric monoidal category enriched over sup-lattices, that is, over *idempotent* commutative monoids with *all* sums (we continue to use $(+, 0)$ for this monoid structure). Any product $A \times B$ in $\mathbf{C}$ is necessarily a *biproduct*, that is, it is also a coproduct and the canonical map $[\langle \mathrm{id}_A, 0\rangle, \langle 0, \mathrm{id}_B\rangle] : A \oplus B \to A \times B$ is an isomorphism. Similarly, every coproduct is a biproduct. Suppose that $\mathbf{C}$ has all *countable* biproducts, and that the monoidal structure distributes over them. We construct a differential structure on the Karoubi envelope $\mathcal{K}(\mathbf{C})$ (idempotent splitting) of $\mathbf{C}$. Recall that this category has as its objects pairs (A, f) where A is an object of $\mathbf{C}$ and $f : A \to A$ is an idempotent, and as its maps $(A, f) \to (B, g)$, those maps $h : A \to B$ from $\mathbf{C}$ such that $h = f; h; g$. This category inherits the monoidal structure, sup-lattice enrichment and biproducts from $\mathbf{C}$.

First, for any object A of $\mathbf{C}$, write $A^{\otimes n}$ to denote the n-fold tensor power of A. The *symmetric* tensor power A^n, if it exists, is the equaliser of the diagram

$$A^{\otimes n} \xrightarrow[\vdots n! \text{ permutations}]{} A^{\otimes n}$$

consisting of all $n!$ permutations from $A^{\otimes n}$ to itself.

In $\mathcal{K}(\mathbf{C})$ we can readily construct symmetric tensor powers, as follows. Given an object A of $\mathbf{C}$, define $\Theta_{A,n} : A^{\otimes n} \to A^{\otimes n}$ to be the sum of the $n!$ permutation maps. Straightforward calculation establishes the following.

Lemma 3. *For any object (A, f) of $\mathcal{K}(\mathbf{C})$, the following diagram is an equalizer.*

$$(A^{\otimes n}, f^{\otimes n}; \Theta_{A,n}) \xrightarrow{f^{\otimes n}; \Theta_{A,n}} (A^{\otimes n}, f^{\otimes n}) \xrightarrow[\vdots n! \text{ permutations}]{} (A^{\otimes n}, f^{\otimes n})$$

Moreover, these equalizer diagrams are preserved by tensor products.

One consequence of this is that there are maps $(A, f)^{m+n} \to (A, f)^m \otimes (A, f)^n$ whose underlying maps are given by $f^{\otimes m+n}; \Theta_{A,m+n}$, as one might expect.

These symmetric tensor powers will allow us to construct a coalgebra modality on $\mathcal{K}(\mathbf{C})$ as the free commutative comonoid. Recall that a *commutative comonoid* in a symmetric monoidal category is an object A together with maps $\Delta : A \to A \otimes A$ and $e : A \to I$ satisfying the obvious commutativity, associativity and unit diagrams; morphisms of comonoids are morphisms between the underlying objects that preserve the comonoid structure. Let $\mathcal{K}^{\otimes}(\mathbf{C})$ be the category of commutative comonoids and comonoid morphisms in $\mathcal{K}(\mathbf{C})$.

Lemma 4. *The forgetful functor $U : \mathcal{K}^{\otimes}(\mathbf{C}) \to \mathcal{K}(\mathbf{C})$ has a right adjoint, whose action on objects takes (A, f) to the biproduct $\bigoplus_{n\in\omega}(A, f)^n$, which we call $!(A, f)$.*

Proof. For any m and n, we have the map

$$\pi_{m+n}; f^{\otimes m+n}; \Theta_{A,m+n} : !(A, f) \longrightarrow (A, f)^m \otimes (A, f)^n.$$

Tupling all these gives us a map $!(A,f) \to \bigoplus_{m,n}(A,f)^m \otimes (A,f)^n$, and by distributivity of tensor over product, this gives a map $\Delta : !(A,f) \to !(A,f) \otimes !(A,f)$. We also have the map $\pi_0 : !(A,f) \to I$. It can readily be verified that these maps give $!(A,f)$ the structure of a comonoid. Moreover, it is the free comonoid on (A,f): if (B,g) is any commutative comonoid and $\alpha : (B,g) \to (A,f)$ any morphism, we construct a comonoid morphism $\alpha^\dagger : (B,g) \to !(A,f)$ as follows. The comultiplication $\Delta^n : (B,g) \to (B,g)^{\otimes n}$ equalizes all permutations, so the composition $\Delta^n; \alpha^{\otimes n}$ does too, yielding a map $(B,g) \to (A,f)^n$. Tupling all these maps gives us the required comonoid map $\alpha^\dagger$, and it is easily checked that this is the unique map such that $\alpha^\dagger; \pi_1 = \alpha$. □

Composing these two adjoint functors yields a comonad $(!, \delta, \epsilon)$ on $\mathcal{K}(\mathbf{C})$.

Lemma 5. *The comonad $(!, \delta, \epsilon)$ is a coalgebra modality. In fact, it is a linear exponential comonad (also known as a storage modality).*

The construction of this comonad along the lines given above follows the recipe in [11], though the use of Karoubi envelope to generate a category possessing the required equalizers seems to be new.

We are now in a position to construct a differential operator on $\mathcal{K}(\mathbf{C})$, making it into a differential category. The differential operator is given by precomposition with the *deriving transformation* $d : (A,f) \otimes !(A,f) \to !(A,f)$ defined as follows. For each n, the map $f^{\otimes n+1}; \Theta_{A,n+1}$ in $\mathbf{C}$ gives us a morphism

$$f^{\otimes n+1}; \Theta_{A,n+1} : (A,f) \otimes (A,f)^n \to (A,f)^{n+1}$$

and hence we obtain maps $\cong; \pi_n; f^{\otimes n+1}; \Theta_{A,n+1} : (A,f) \otimes !(A,f) \to (A,f)^{n+1}$ where $\cong$ is the distributivity map. Tupling all these gives us a morphism $(A,f) \otimes !(A,f) \to \bigoplus_n (A,f)^{n+1}$, and finally pairing this with $0 : (A,f) \otimes !(A,f) \to I$ gives the required map.

Theorem 6. *With the structure described above, $\mathcal{K}(\mathbf{C})$ is a sup-lattice-enriched differential category, and the Kleisli category $\mathcal{K}_!(\mathbf{C})$ a cpo-enriched Cartesian differential category. If $\mathbf{C}$ is monoidal closed (in the sup-lattice-enriched sense) then $\mathcal{K}_!(\mathbf{C})$ is a cpo-enriched Cartesian-closed differential category.*

Proof. That $\mathcal{K}(\mathbf{C})$ is a differential category is lengthy but straightforward to check. Sup-lattice enrichment follows directly from that of $\mathbf{C}$. The fact that $\mathcal{K}_!(\mathbf{C})$ is Cartesian differential follows from Proposition 3.2.1 of [4]. The Cartesian closure of $\mathcal{K}_!(\mathbf{C})$ is a well-known fact about linear exponential comonads. For the cpo-enrichment, it is enough to observe that the passage from $\alpha : !(A,f) \to (B,g)$ to $\alpha^\dagger : !(A,f) \to !(B,g)$ preserves directed suprema. □

Even when $\mathbf{C}$ is not monoidal closed, it is still possible to arrive at a Cartesian closed differential category when there are enough exponentials: if $\mathbf{C}$ has all exponentials $A \multimap R$ for some fixed object R, then the full subcategory of $\mathcal{K}_!(\mathbf{C})$ consisting of such R-exponentials is Cartesian closed, and also possesses a weak distributive coproduct structure, the "lifted sum" $\Sigma_{i\in I} A_i = (\bigoplus_{i\in I}(!A_i \multimap R)) \multimap R$. In particular, $\bigoplus_{n\in\omega} R \multimap R$ is a weak natural numbers object.

To apply the construction above, we need a sup-lattice enriched symmetric monoidal category with countable distributive biproducts. Such categories can readily be manufactured via a series of free constructions.

Beginning with a symmetric monoidal category, one can construct its *sup-lattice-completion* as the category with the same objects, but whose maps $A \to B$ are sets of maps in the original category (cf. [9] VIII.2 exercise 5). This is a sup-lattice enriched category, with joins of maps given by unions, and monoidal structure inherited from the original category; closed structure is also inherited, if it exists. We denote the sup-lattice completion of a category $\mathbf{C}$ by $\mathbf{C}^+$.

Given a sup-lattice-enriched symmetric monoidal category, its *biproduct completion* (cf. [9] VIII.2 exercise 6) has as objects indexed sets $\{A_i \mid i \in I\}$ of objects in the original category, and as morphisms $\{A_i \mid i \in I\} \to \{B_j \mid j \in J\}$ *matrices* of morphisms, that is, for each i, j, a morphism $A_i \to B_j$. Composition is (potentially infinite) matrix multiplication; the infinite sums required for composition are the reason we require sup-lattice enrichment. The biproduct of a set of objects is given by the disjoint union of families. We write $\mathbf{BP}(\mathbf{C})$ for the biproduct completion of a category $\mathbf{C}$.

We will be interested in some categories which arise by performing these two constructions in sequence. Given a category $\mathbf{C}$, we denote by $\mathbf{FamRel}(\mathbf{C})$ the category whose objects are families $\{A_i \mid i \in I\}$ of objects of $\mathbf{C}$, and whose morphisms $\{A_i \mid i \in I\} \to \{B_j \mid j \in J\}$ are given by sets of triples (i, j, f) where $i \in I$, $j \in J$ and $f : A_i \to B_j$ in $\mathbf{C}$. Note that for a given i and j there may be no such triples in a morphism, or one, or many. It is easy to check that $\mathbf{FamRel}(\mathbf{C})$ is isomorphic to the category $\mathbf{BP}(\mathbf{C}^+)$.

A simple but central example begins with the terminal category $\mathbf{1}$. $\mathbf{FamRel}(\mathbf{1})$ is the category $\mathbf{Rel}$ of sets and relations. On the image of $\mathbf{Rel}$ in $\mathcal{K}(\mathbf{Rel})$, ! is the finite-multiset comonad, and we therefore find $\mathbf{MRel}$ embedded in $\mathcal{K}_!(\mathbf{Rel})$ as a sub-Cartesian-differential-category.

5 Deconstructing Categories of Games

In this section we apply some of the constructions developed above to reconstruct $\mathbf{G}^{\otimes}$ and discover its differential structure as an instance of our construction. We begin by defining a new category $\mathbf{EG}$ of *exhausting games*, to which we will apply our constructions, eventually arriving at a Cartesian closed differential category containing $\mathbf{G}^{\otimes}$ as a subcategory. We then introduce a refined category of exhausting games, $\mathbf{EG}_{\sim}$, and again apply our constructions to obtain a model of Resource PCF with the finite definability property.

Given a finite arena A, a *path* is a non-repeating enumeration of all moves, respecting the order given by the edge relation in the arena — that is, a traversal of the forest — such that the first move is by O and moves alternate polarity thereafter. Note that every move in a path has a unique justifier earlier in the path. An *exhausting strategy* on A is a set of even-length paths that satisfy P-visibility; if A has an odd number of moves, the only strategy is the empty set. The category $\mathbf{EG}$ has finite O-rooted arenas as objects and exhausting strategies

on $A^{\perp} \uplus B$ as maps from A to B, with composition and identities as usual. Again, disjoint union of arenas gives a monoidal structure; and if B has a single root, then the arena $A \multimap B$ is an exponential, so **EG** has all R-exponentials, where R is the arena with a single move belonging to O.

It is clear that **EG** is sup-lattice enriched: unions of strategies are strategies, and composition preserves unions. We may then form its biproduct completion, to obtain the structure we require to construct a differential category as in Section 4. We write $\mathcal{K}(\mathbf{BP}(\mathbf{EG}))$ for the differential category so constructed, and $\mathcal{K}_!(\mathbf{BP}(\mathbf{EG}))$ for its Kleisli category, which is a Cartesian differential category. The full subcategory of R-exponentials is Cartesian closed and has a weak natural numbers object. We now show how to recover $\mathbf{G}^{\otimes}$ as a subcategory of this.

Let A be any QA-arena, and consider a non-repeating (justified) sequence s of pairs (a, n) where a is a move of A and n is a natural number. Taking left projection on such a sequence gives a justified sequence in A, which we call $\hat{s}$; we say that s is a tagging of $\hat{s}$, and write $\mathsf{tcomp}(A)$ for the set of all taggings of complete well-opened plays in A.

Let $s \in \mathsf{tcomp}(A)$ be a tagging of the complete play $\hat{s}$. We can define an arena $\|s\|$ whose moves are the elements of s and whose edge relation is precisely the justification structure of s. Thus s becomes a path of $\|s\|$. If t is a tagged sequence such that $\hat{t} = \hat{s}$, there is an isomorphism between the moves of $\|s\|$ and $\|t\|$ which maps the n-th move of s to the n-th move of t. The free monoid extension induces an isomorphism ϕ between paths in $\|s\|$ and those in $\|t\|$, and in turn an isomorphism in **EG**, given by the strategy $\phi_{\|s\|,\|t\|} = \{u \in \|s\|^{\perp} \uplus \|t\| \mid \hat{u} \in \mathrm{id}_A, \phi(u \upharpoonright \|s\|) = u \upharpoonright \|t\|\}$. Let A^* be the family of arenas $\{\|s\| \mid s \in \mathsf{tcomp}(A)\}$. We define a morphism $\phi_A : A^* \to A^*$ in the biproduct completion of **EG** as the "matrix" with entries given by

$$(\phi_A)_{s,t} = \begin{cases} \phi_{\|s\|,\|t\|}, & \text{if } \hat{s} = \hat{t} \\ \emptyset, & \text{otherwise.} \end{cases}$$

Our embedding of $\mathbf{G}^{\otimes}$ in $\mathcal{K}_!(\mathbf{BP}(\mathbf{EG}))$ maps an arena A to (A^*, ϕ_A). The action on morphisms is slightly trickier to describe; we begin by analysing complete, well-opened plays in $A^{\perp} \uplus B$. Such a play consists of an interleaving of a complete, well-opened play in B with a number of complete, well-opened plays in A. Suppose the play s is an interleaving of plays $s_1, \ldots, s_n$ in A and s' in B. Let u be any tagging of s, yielding taggings $u_1, \ldots, u_n$ and u' of the projections s_1, $\ldots$, s_n and s'. This tagging induces a morphism $(A^*)^{\otimes n} \to B^*$ in the biproduct completion of **EG**, as follows.

The family $(A^*)^{\otimes n}$ consists of arenas $\|s'_1\| \otimes \cdots \otimes \|s'_n\|$ for $s'_i \in \mathsf{tcomp}(A)$. If $\|s'_i\| = \|u_i\|$ for each i and $\|s'\| = \|u'\|$, then the singleton u is a morphism

$$\{u\} : \|s'_1\| \otimes \cdots \otimes \|s'_n\| \to \|s'\|$$

and hence, taking all such taggings and inserting empty strategies everywhere else in the matrix, we have a morphism $(A^*)^{\otimes n} \to B^*$.

Given a map $\sigma : A \to B$ in $\mathbf{G}^{\otimes}$, let σ_n be the set of well-opened plays in σ with n initial A-moves. Each $s \in \sigma_n$ induces a morphism $(A^*)^{\otimes n} \to B^*$ in **EG**

as above, so we can define $\Phi(\sigma_n) : (A^*)^{\otimes n} \to B^*$ to be the sum of all these morphisms. The copairing of the $\Phi(\sigma_n)$ gives us a map

$$[\Phi(\sigma_n) \mid n \in \omega] : \bigoplus_{n\in\omega} (A^*)^{\otimes n} \to B^*.$$

By construction, for each $\Phi(\sigma_n)$ we have $\Phi(\sigma_n) = \phi_A^{\otimes n}; \Theta_{A^*,n}; \Phi(\sigma_n); \phi_B$, so $[\Phi(\sigma_n) \mid n \in \omega]$ is a morphism from $!(A^*, \phi_A)$ to (B^*, ϕ_B) in $\mathcal{K}(\mathbf{BP}(\mathbf{EG}))$.

Proposition 7. *The construction defined above gives a full and faithful product-preserving functor from* $\mathbf{G}^{\otimes}$ *to* $\mathcal{K}_!(\mathbf{BP}(\mathbf{EG}))$.

Thus the category $\mathbf{G}^{\otimes}$ is rich enough to interpret Resource PCF. Indeed, it is richer, as the model of EIA demonstrates, and the model of Resource PCF in $\mathbf{G}^{\otimes}$ is far from being fully abstract. We now develop a model of Resource PCF which possesses the finite definability property, by applying our general construction to a refined category of exhausting games.

Let A be any arena. We define an equivalence relation $\sim$ on the paths of A as the smallest equivalence relation such that $s \cdot o \cdot p \cdot o' \cdot p' \cdot t \sim s \cdot o' \cdot p' \cdot o \cdot p \cdot t$ where o, o' are O-moves and p, p' are P-moves. We call a path *safe* if, whenever $s = s' \cdot o \cdot p \cdot o' \cdot p' \cdot t$ and o justifies p', p' justifies o'. The $\sim$ relation captures a notion of causal independence similar to that of Melliès [12], and allows us to refine our games model to obtain definability for Resource PCF.

A *$\sim$-strategy* σ on an arena A is a set of safe paths that is $\sim$-closed, that is, if $s \in \sigma$ and $s \sim t$ then $t \in \sigma$. A $\sim$-strategy σ is *deterministic* if it is non-empty, and the longest common prefix of any $s, t \in \sigma$ has even length.

Lemma 8. *If σ is a $\sim$-strategy and $s \in \sigma$ then s satisfies P-visibility.*

Given a path s in an arena A, write $\widetilde{s}$ for the equivalence class of s under $\sim$.

Lemma 9. *For any safe path s of A, $\widetilde{s}$ is a deterministic $\sim$-strategy, and any deterministic $\sim$-strategy is of the form $\widetilde{s}$ for some safe path s.*

We can now build two categories: $\mathbf{EG}_\sim$ has O-rooted arenas as objects and deterministic $\sim$-strategies on $A^\perp \uplus B$ as maps $A \to B$; $\mathbf{EG}_\sim^+$ has the same objects, but its maps $A \to B$ are arbitrary $\sim$-strategies on $A^\perp \uplus B$. $\mathbf{EG}_\sim^+$ is therefore the subcategory of $\mathbf{EG}$ consisting of $\sim$-closed strategies.

We are ready to construct a Cartesian differential category, starting with $\mathbf{EG}_\sim$. The first step is to take its sup-lattice completion; a consequence of Lemma 9 is that this is exactly $\mathbf{EG}_\sim^+$, justifying our choice of nomenclature.

Lemma 10. $\mathbf{EG}_\sim^+$ *is the sup-lattice completion of* $\mathbf{EG}_\sim$.

Nevertheless, it is convenient to take the first two steps of the construction together, working with $\mathbf{FamRel}(\mathbf{EG}_\sim)$. Our construction gives us a comonad ! on $\mathcal{K}(\mathbf{FamRel}(\mathbf{EG}_\sim))$, such that the Kleisli category $\mathcal{K}_!(\mathbf{FamRel}(\mathbf{EG}_\sim))$ is a Cartesian differential category. Though $\mathbf{EG}_\sim$ is not monoidal closed, it has all R-exponentials, so the full subcategory of $\mathcal{K}_!(\mathbf{FamRel}(\mathbf{EG}_\sim))$ comprising the arenas with a single root is a Cartesian closed differential category.

As before, we may give a direct definition of a category of games which is a sub-Cartesian-closed-differential-category of this one. Let $\mathbf{G}_\sim$ be the subcategory of $\mathbf{G}$ consisting of $\sim$-closed strategies. Again taking the subcategory of

comonoid homomorphisms, we arrive at a Cartesian closed differential category $\mathbf{G}^{\otimes}_{\sim}$. Just as in Sec. 5, we can define a full and faithful functor from $\mathbf{G}^{\otimes}_{\sim}$ into $\mathcal{K}_!(\mathbf{FamRel}(\mathbf{EG}_\sim))$ which preserves the Cartesian closed differential structure.

6 Analysing Models of Resource PCF

Our constructions show that each of $\mathcal{K}_!(\mathbf{FamRel}(\mathbf{EG}_\sim))$, $\mathcal{K}_!(\mathbf{BP}(\mathbf{EG}))$ and $\mathcal{K}_!(\mathbf{FamRel}(\mathbf{1}))$ is a cpo-enriched differential Cartesian category with enough exponentials to interpret Resource PCF; and indeed we have identified full subcategories $\mathbf{G}^{\otimes}$, $\mathbf{G}^{\otimes}_{\sim}$ and $\mathbf{MRel}$ which are cpo-enriched Cartesian closed differential categories containing all the objects needed to interpret Resource PCF soundly. However, for $\mathbf{G}^{\otimes}_{\sim}$ and $\mathbf{MRel}$, there is more to be said.

Consider those arenas for which there exists a Q/A-labelling such that every question enables a unique answer — this is a constraint on the shapes of the trees, rather than additional structure. We write $\mathbf{EG}^{QA}_{\sim}$ for the full subcategory of $\mathbf{EG}_\sim$ consisting of such arenas, and note that $\mathbf{G}^{\otimes}_{\sim}$ embeds in $\mathcal{K}_!(\mathbf{FamRel}(\mathbf{EG}^{QA}_{\sim}))$ by construction.

Lemma 11. *For every such arena, the set of safe paths is non-empty.*

Proof. We construct a safe path by induction on partial paths. Begin with any root node. Having constructed a partial path s, consider the pending question in s. If it enables any questions that do not appear in s, extend s with one of them. Otherwise, extend s with the unique answer of the pending question. If there is no pending question, extend s with any question enabled by one of the answers in the P-view of s, if one exists, or an unplayed root node, if one exists. If no such moves exist, s is a safe path. □

Corollary 12. *The unique functor $\top : \mathbf{EG}^{QA}_{\sim} \to \mathbf{1}$ is full. (This amounts to the fact that the set of safe paths of $A^\perp \uplus B$ is non-empty.)*

This full functor extends through our constructions to a full functor from $\mathcal{K}_!(\mathbf{FamRel}(\mathbf{EG}^{QA}_{\sim}))$ to $\mathcal{K}_!(\mathbf{FamRel}(\mathbf{1}))$. Moreover, the only idempotents we make use of in the Karoubi envelope have the form $\sum_{f\in G} f$ where G is some group of automorphisms. In the case of $\mathbf{Rel}$, these idempotents are equivalence relations, and an object $(A, \simeq)$ in $\mathcal{K}(\mathbf{Rel})$ is isomorphic to $(A/\simeq, \mathrm{id}_A)$. The part of the Karoubi envelope that is used in our constructions is therefore equivalent to $\mathbf{Rel}$ itself, with the comonad being the usual finite-multiset comonad, and the Kleisli category being $\mathbf{MRel}$. We therefore obtain a full functor from $\mathbf{G}^{\otimes}_{\sim}$ to $\mathbf{MRel}$ which preserves all the relevant structure.

This functor may be described concretely as follows. Given a complete justified sequence s on a QA-arena, write $|s|$ for the underlying multiset of moves of s, partially ordered by the justification relation. The functor sends an arena A to the set of all such pomsets, which we call the *positions* of A. If s is a well-opened complete justified sequence on $A^\perp \uplus B$, $|s|$ is a pair consisting of a multiset of positions of A and a position of B. The functor sends a map $A \to B$ to the set of positions of its sequences. This is essentially the "time-forgetting" map of [2], which here is functorial because of $\sim$-closure.

Theorem 13. *The models of Resource PCF in* $\mathbf{G}^{\otimes}_{\sim}$ *and* **MRel** *have the finite definability property, and the model in* **MRel** *is fully abstract.*

Proof. For $\mathbf{G}^{\otimes}_{\sim}$, a straightforward induction on the size of strategies, following the steps in the definability proof for the innocent strategy model of PCF. $\sim$-closure ensures that strategies are insensitive to the order in which O-moves are made. Definability for **MRel** follows from the fullness of the positional collapse of $\mathbf{G}^{\otimes}_{\sim}$ onto **MRel**. For full abstraction of **MRel**, let M and N be closed terms of type A. If $[\![M]\!] \not\subseteq [\![N]\!]$, there is some $a \in [\![M]\!] \setminus [\![N]\!]$. By finite definability, the relation $\{(a, 0)\} : A \to \mathtt{nat}$ is the denotation of some term $x : A \vdash C(\!|x|\!) : \mathtt{nat}$. Therefore $[\![C(\!|M|\!)]\!] = [\![\mathrm{zero}]\!]$ while $[\![C(\!|N|\!)]\!] = \emptyset$, so $C(\!|M|\!) \Downarrow$ but $C(\!|N|\!) \not\Downarrow$. □

Acknowledgements. Research supported in part by NWO Project 612.000.936 CALMOC and by UK EPSRC grant EP/HO23097.

References

1. Abramsky, S., McCusker, G.: Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In: O'Hearn, P.W., Tennent, R.D. (eds.) Algol-like Languages, vol. 2, pp. 297–329. Birkhaüser, Basel (1997)
2. Baillot, P., Danos, V., Ehrhard, T., Regnier, L.: Timeless games. In: Gottlob, G., Grandjean, E., Seyr, K. (eds.) CSL 1998. LNCS, vol. 1584, pp. 56–77. Springer, Heidelberg (1999)
3. Blute, R.F., Cockett, J.R.B., Seely, R.A.G.: Differential categories. Mathematical Structures in Comp. Sci. 16, 1049–1083 (2006)
4. Blute, R.F., Cockett, J.R.B., Seely, R.A.G.: Cartesian differential categories. Theory and Applications of Categories 22(23), 622–672 (2009)
5. Bucciarelli, A., Ehrhard, T., Manzonetto, G.: Categorical models for simply typed resource calculi. ENTCS 265, 213–230 (2010); Proc. of MFPS 2010
6. Ehrhard, T., Regnier, L.: The differential lambda-calculus. Theor. Comput. Sci. 309, 1–41 (2003)
7. Harmer, R.: Games and full abstraction for nondeterministic languages. PhD thesis, University of London (1999)
8. Harmer, R., McCusker, G.: A fully abstract game semantics for finite nondeterminism. In: LICS 1999, pp. 422–430 (1999)
9. Mac Lane, S.: Categories for the working mathematician. Springer, Berlin (1971)
10. McCusker, G.: Games and full abstraction for a functional metalanguage with recursive types. Distinguished Dissertations in Comp. Sci. Springer, Heidelberg (1998)
11. Melliès, P.-A., Tabareau, N., Tasson, C.: An explicit formula for the free exponential modality of linear logic. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 247–260. Springer, Heidelberg (2009)
12. Melliès, P.-A.: Asynchronous games 2: the true concurrency of innocence. Theor. Comput. Sci. 358, 200–228 (2006)
13. Pagani, M., Tranquilli, P.: Parallel reduction in resource lambda-calculus. In: Hu, Z. (ed.) APLAS 2009. LNCS, vol. 5904, pp. 226–242. Springer, Heidelberg (2009)

Nondeterminism Is Essential in Small 2FAs with Few Reversals

Christos A. Kapoutsis

LIAFA – Université Paris VII

Abstract. On every n-long input, every two-way finite automaton (2FA) can reverse its head $O(n)$ times before halting. A 2FA *with few reversals* is an automaton where this number is only $o(n)$. For every h, we exhibit a language that requires $\Omega(2^h)$ states on every deterministic 2FA with few reversals, but only h states on a nondeterministic 2FA with few reversals.

1 Introduction

A long-standing open question in the theory of computation, posed already in the 70s [11,10], is whether every two-way nondeterministic finite automaton (2NFA) has a deterministic equivalent (2DFA) with at most polynomially more states.

The answer is conjectured to be negative. Indeed, this has been confirmed in several special cases: for automata that are *single-pass* (halting upon reaching an endmarker [11]) or *sweeping* (reversing only on endmarkers [12,9]) or *almost oblivious* (exhibiting $o(n)$ distinct trajectories on n-long inputs [5]) or *moles* (exploring the implicit configuration graph [7]). However, for *unary* automata a non-trivial upper bound is known: the simulating 2DFA need never be more than quasi-polynomially larger [2]. We also know that the final answer, both for general and for unary alphabet, may have implications for the old question whether nondeterminism is essential in space-bounded Turing machines [1,3,8].

Here we confirm the general conjecture in yet another special case: for automata that reverse their input head (anywhere on the tape, but) only $o(n)$ times on every n-long input before halting. These '2FAs *with few reversals*' stand very naturally between sweeping 2FAs, which perform only $O(1)$ reversals and only on the endmarkers, and general 2FAs, which perform $O(n)$ reversals (cf. Lemma 1).

Theorem 1. For every h, there is a language that requires $\Omega(2^h)$ states on every 2DFA *with few reversals* but only h states on a 2NFA *with few reversals*.

Here, the family of witness languages is ONE-WAY LIVENESS [10] (as usual [12,5,7]) and the h-state 2NFAs are actually one-way (so that their 'few' reversals are in fact 'zero'). So, this can be seen as a generalization of the lower bound of [12].

Given Theorem 1, two questions arise. First, *does the theorem really generalize* [12], or can it perhaps follow from it by proving that the gap from few-reversal

* Research funded by a Marie Curie Intra-European Fellowship (PIEF-GA-2009-253368) within the European Union Seventh Framework Programme (FP7/2007-2013).

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 198–209, 2011.

to sweeping 2DFAs is only polynomial? Second, *does the full conjecture really generalize the theorem*, or can it perhaps follow from it by proving that the gap from general to few-reversal 2DFAs is only polynomial? We answer affirmatively.

Theorem 2. For every h, there is a language that requires $2^{\Omega(h)}$ states on every *sweeping* 2DFA but only $O(h)$ states on a 2DFA *with* 2 *reversals*.

Theorem 3. For every h, there is a language that requires $\Omega(2^h)$ states on every 2DFA *with few reversals* but only $O(h^2)$ states on a *general* 2DFA.

Note that Theorems 1-3 answer Hromkovič's Research Problems 2 and 3 from [4].

We consider the second half of Theorem 1 (upper bound) known, and leave the proofs of Theorems 2 and 3 for longer versions of this report. We thus prove only the first part of Theorem 1 (lower bound). We work as in [12]. After fixing notation (Sect. 2), we define *generic strings* (Sect. 3.1), study their *blocks* (Sect. 3.2), build a family of *hard instances* using such blocks (Sect. 4.1), and apply the linear algebra bound on vectors derived from these instances (Sect. 4.2).

2 Preparation

Let $[n] := \{0, \ldots, n-1\}$. For A a set, $\overline{A}$ is the complement and $\mathrm{id}_A : A \to A$ the identity on A. For $f : A \rightharpoonup B$ a partial function, $X \subseteq A$, and $b \in B$, we define $f[X] := \{f(a) \mid a \in X \ \&\ f(a)\,\text{defined}\}$ and $f^{-1}(b) := \{a \in A \mid f(a) = b\}$. If in addition $g : B \rightharpoonup C$, then the composition $f \circ g : A \rightharpoonup C$ is defined on a iff both $f(a)$ and $g(f(a))$ are defined. If $A = B$, then f^n is the n-fold composition of f with itself, and $f \leq g$ means that $f(a)\,\text{defined} \Longrightarrow g(a)\,\text{defined}\ \&\ g(a) = f(a)$.

Fact 1. *The relation* $\leq$ *is a partial order on* $\mathcal{F}_A := \{f \mid f : A \rightharpoonup A\}$, *and* id_A *is maximal in it. Moreover,* $g \leq g' \Longrightarrow f \circ g \leq f \circ g'$ *for all* $f,g,g' \in \mathcal{F}_A$.

For z a string, $|z|$ and z_j denote its length and its j-th symbol ($1 \leq j \leq |z|$). A (*promise*) *problem* over Σ is any pair $(L, \tilde{L})$ of disjoint subsets of Σ^*. A machine *solves* $(L, \tilde{L})$ if it accepts all $w \in L$ but no $w \in \tilde{L}$. If $\tilde{L} = \overline{L}$, then L is a *language*.

Liveness. For $h \geq 1$, the alphabet $\Sigma_h := \{\text{all } G \subseteq [h] \times [h]\}$ is all two-column directed graphs with h nodes per column and only rightward arrows (Fig. 1a). An n-long $z \in \Sigma_h^*$ is viewed as an $(n+1)$-column graph (without arrow directions, for simplicity); its *connectivity* is $\xi := \{(a, b) \mid \text{there is an } n\text{-arrow path from node } a \text{ of column 0 to node } b \text{ of column } n\}$; if $\xi = \emptyset$, then z is *dead*, otherwise it is *live*. We define $\mathrm{OWL}_h = \text{ONE-WAY LIVENESS}_h := \{z \in \Sigma_h^* \mid z \text{ is live}\}$ [10].

Machines. A *two-way deterministic finite automaton* (2DFA) is any tuple $M = (Q, \Sigma, \delta, q_s, q_a, q_r)$, where Q is a set of *states*, Σ an *alphabet*, $q_s, q_a, q_r \in Q$ are the *start*, *accept*, and *reject* states, and $\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to Q \times \{\mathtt{l}, \mathtt{r}\}$ is the (total) *transition function*, for $\vdash, \dashv \notin \Sigma$ two endmarkers and $\mathtt{l}$,$\mathtt{r}$ the two directions. An input $w \in \Sigma^*$ is presented to M endmarked, as $\vdash w \dashv$. Computation starts at q_s and on $\vdash$. In each step, the next state and head move are derived from δ and

the current state and symbol. Endmarkers are never violated, except for $\dashv$ if the next state is q_a or q_r; i.e., $\delta(\cdot,\vdash)$ is always of the form $(\cdot,\mathtt{r})$, and $\delta(\cdot,\dashv)$ is always $(q_a,\mathtt{r})$ or $(q_r,\mathtt{r})$ or of the form $(\cdot,\mathtt{l})$. So, the computation either loops, or falls off $\dashv$ into q_r, or falls off $\dashv$ into q_a. In the last case, we say M *accepts* w.

In general, the *computation of M from state p on the j-th symbol of string z*, denoted $\text{COMP}_{M,p,j}(z)$, is the longest sequence $c=((q_t,j_t))_{0\le t<m}$ with $0<m\le\infty$, $(q_0,j_0)=(p,j)$, and every next (q_t,j_t) derived from the previous one via δ and z in the natural way. We say (q_t,j_t) is the *t-th point* and m the *length*. If $m=\infty$ then c *loops*; otherwise, $j_{m-1}=0$ or $|z|{+}1$ and c *hits left* or *hits right*, respectively, *into* q_{m-1} (Fig. 1b). We say $c'=((q'_t,j'_t))_{0\le t<m'}$ *parallels* c if it is a 'shifted copy' of it: $m'=m$ and $q'_t=q_t$ & $j'_t=j_t+j_*$ for some j_* and all t.

The L-*computation of M from p on z* is $\text{LCOMP}_{M,p}(z):=\text{COMP}_{M,p,1}(z)$ and is called a LR-*traversal*, L-*turn*, or L-*loop*, depending on whether it hits right, hits left, or loops. Similarly, the R-computation $\text{RCOMP}_{M,p}(z):=\text{COMP}_{M,p,|z|}(z)$ is a RL-*traversal*, R-*turn*, or R-*loop*. Two L-/R-computations *resemble* each other if they share the same first state, type (L/R-turn/loop, LR/RL-traversal), and last state (if it exists). The (*full*) *computation of M on $w\in\Sigma^*$* is $\text{COMP}_M(w):=\text{LCOMP}_{M,q_s}(\vdash w\dashv)$. Hence, M accepts w iff $\text{COMP}_M(w)$ hits right into q_a.

For $w=uzv$, the *decomposition of $c:=\text{COMP}_M(w)$ by z* is the unique sequence $c_0,c_1,\dots$ of computations, called *segments*, derived by splitting c wherever it enters or exits z (for each point (q_t,j_t) produced by crossing the u-z or z-v boundary, replace (q_t,j_t) by two copies of it and split between the copies). Note that every c_i for even i (resp., odd i) is a computation on $\vdash u$ or $v\dashv$ (resp., z), and c halts iff there exists a last segment c_m and m is even and c_m falls off $\dashv$.

We say M is *nondeterministic* (2NFA) if δ maps $Q\times(\Sigma\cup\{\vdash,\dashv\})$ to the *powerset* of $Q\times\{\mathtt{l},\mathtt{r}\}$. Then every $\text{COMP}_{M,p,j}(z)$ is a *set* of computations, and M accepts w iff some $c\in\text{COMP}_M(w)$ hits right into q_a.

A *reversal* in a computation c is any point (q_t,j_t) whose predecessor and successor exist and lie on the same side with respect to it (Fig. 1b): $t\ne 0,m{-}1$ and either $j_{t-1},j_{t+1}<j_t$ (*backward* reversal) or $j_t<j_{t-1},j_{t+1}$ (*forward* reversal). We write $r(c)$ for the total number of reversals in c. Note that $0\le r(c)\le\infty$, and $r(c)=\infty$ iff c is looping. For $n\ge 0$, we write $r_M(n)$ for the maximum $r(c)$ over all full computations c of M on n-long inputs.

Lemma 1. *For every s-state* 2DFA *M and every length n, either $r_M(n)=\infty$ or $r_M(n)$ is even and at most $(s-1)(n+2)$.*

3 Building Hard Instances

Hard instances for 2DFAs are built in three stages. We start with *generic strings*, which buy us some basic stability in the machine's behavior. We then use generic strings to build *blocks*, where we draw a set of requirements for how the machine may compute. Finally, in order to force her to meet these requirements, we iterate the blocks into long strings, and ask her to decide correctly there. This general strategy is from [12]. Its instantiation here for 2DFAs improves on [7, §3].

From now on, we fix 2DFA $M{=}(Q,\Sigma,\delta,q_s,q_a,q_r)$ and drop it from subscripts.

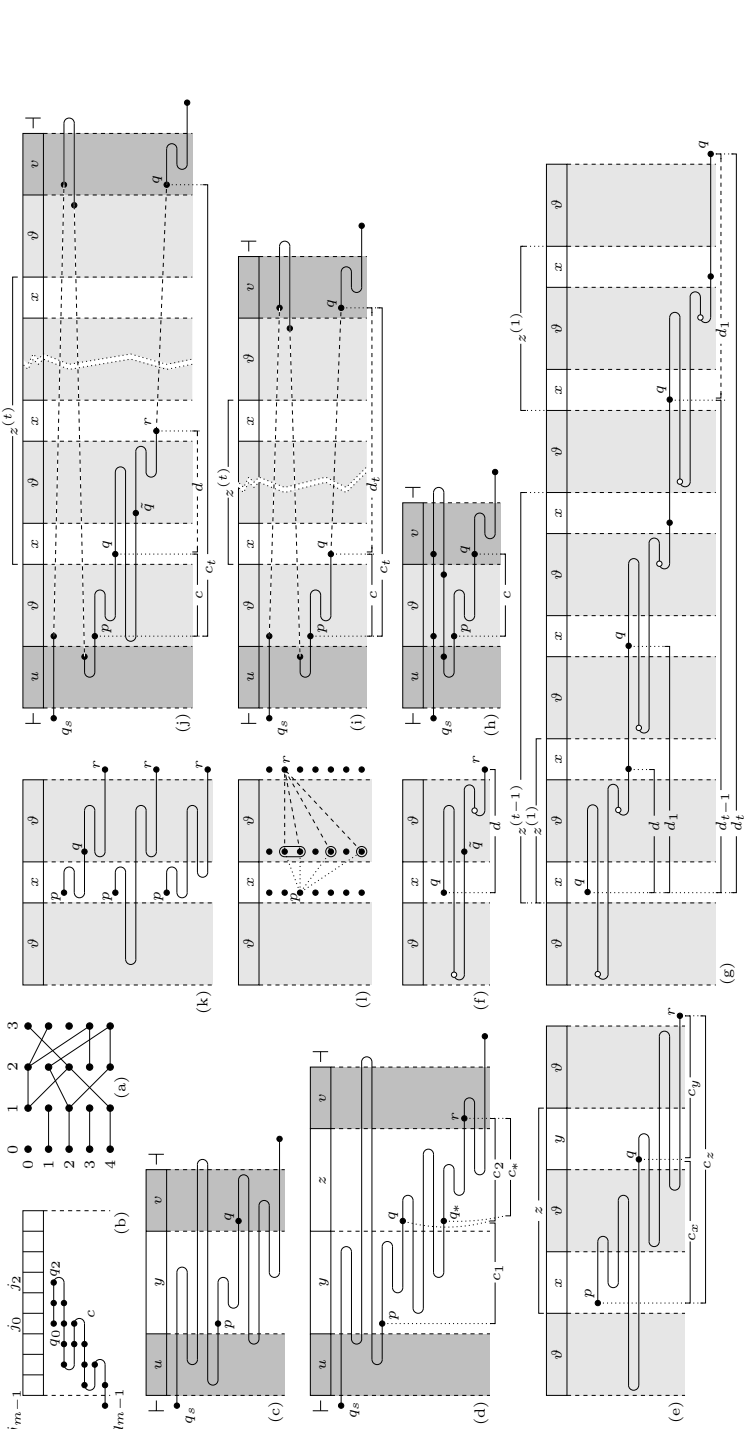


Fig. 1. (a) A string of 3 symbols over Σ_5, drawn as an indexed 4-column undirected graph; here $\xi = \{(2,4),(4,0)\}$. (b) A left-hitting computation c with $m = 15$, $r(c) = 5$; points 2, 8, 12 are backward reversals, points 6, 11 are forward reversals. (c) Computing on uyv. (d) Computing on $uyzv$. (e) Computing on overlapping blocks $\vartheta x\vartheta, \vartheta y\vartheta$. (f) Witnessing $\alpha_x(q) = r$, with two forward reversals. (g) Induction for proving Lemma 5 (here $l = 2$, $t = 3$). (h) Witnessing $q \in A$. (i) Proving Lemma 5. (j) Proving Fact 6. (k) Three right-hitting $\textsc{comp}_{p,|\vartheta|+1}(\vartheta x\vartheta)$: one simple (top), two non-simple. (l) Understanding S_p: each column is a copy of Q; circles and dashed edges represent the q for which $\textsc{lcomp}_q(\vartheta)$ hits right into r; dotted edges represent the $\delta_{\mathrm{LR}}(p,x,q)$ that are being summed.

3.1 Generic Strings

For each $y \in \Sigma^*$, consider all states that can be produced by LR-traversals of y inside full computations of M (Fig. 1c), called the LR-*outcomes* of y:

$$Q_{\text{LR}}(y) := \big\{ q \in Q \mid \text{there exist } p \text{ and } u,v \text{ such that } \text{LCOMP}_p(y) \text{ appears in } \text{COMP}(uyv) \ \&\ \text{hits right into } q \big\}, \quad (1)$$

where a computation on y 'appears in $\text{COMP}(uyv)$' if it parallels one of the odd-indexed segments in the decomposition of $\text{COMP}(uyv)$ by y.

We now consider any extension yz of y and compare $Q_{\text{LR}}(yz)$ with $Q_{\text{LR}}(y)$ and $Q_{\text{LR}}(z)$. For the first comparison, we define a partial function $\alpha_{y,z} : Q_{\text{LR}}(y) \rightharpoonup Q$ as follows (Fig. 1d): for each $q \in Q_{\text{LR}}(y)$, examine $\text{COMP}_{q,|y|+1}(yz)$; if it hits right into some state r, set $\alpha_{y,z}(q) := r$; if it hits left or loops, leave $\alpha_{y,z}(q)$ undefined.

Fact 2a. *For all* $y,z \in \Sigma^*$: $Q_{\text{LR}}(yz) \subseteq \alpha_{y,z}[Q_{\text{LR}}(y)] \cap Q_{\text{LR}}(z)$.

Proof. Let $r \in Q_{\text{LR}}(yz)$. Then there exist p and u,v such that $c := \text{LCOMP}_p(yz)$ appears in $\text{COMP}(uyzv)$ and hits right into r (Fig. 1d). We know c crosses the y-z boundary at least once. Let q and q_* be the states right after the first crossing and after the last crossing, respectively. The prefix of c up to the first crossing is $c_1 := \text{LCOMP}_p(y)$ and hits right into q, while the remaining suffix is $c_2 := \text{COMP}_{q,|y|+1}(yz)$ and hits right into r. The suffix of c after the last crossing is $c_* = \text{LCOMP}_{q_*}(z)$ and hits right into r. Now, c_1 is a LR-traversal of y that appears in $\text{COMP}(uyzv)$ and produces q, so $q \in Q_{\text{LR}}(y)$. By this and c_2, we know $\alpha_{y,z}(q) = r$. Therefore, $r \in \alpha_{y,z}[Q_{\text{LR}}(y)]$. Moreover, c_* is a LR-traversal of z that appears in $\text{COMP}(uyzv)$ and produces r. Therefore, $r \in Q_{\text{LR}}(z)$. □

Symmetrically, we let the set $Q_{\text{RL}}(y)$ of RL-*outcomes* of y be all states producible by RL-traversals of y inside full computations of M. Then $\beta_{z,y} : Q_{\text{RL}}(y) \rightharpoonup Q$ is introduced so that $\beta_{z,y}(q)$ is r, if $\text{COMP}_{q,|z|}(zy)$ hits left into r, or undefined, if the computation loops or hits right. Then a fact symmetric to Fact 2a holds.

Fact 2b. *For all* $y,z \in \Sigma^*$: $Q_{\text{RL}}(zy) \subseteq \beta_{z,y}[Q_{\text{RL}}(y)] \cap Q_{\text{RL}}(z)$.

By the first inclusion of Fact 2a, we know $|Q_{\text{LR}}(y)| \geq |Q_{\text{LR}}(yz)|$. Similarly, Fact 2b implies $|Q_{\text{RL}}(zy)| \leq |Q_{\text{RL}}(y)|$. Hence, extending a string in either direction can never increase the respective number of outcomes. Thus, sufficiently long extensions will minimize this number. Such extensions are called *generic strings*.

Definition. *Let* $T \subseteq \Sigma^*$ *be arbitrary. A string* $y \in T$ *is* LR-generic (for M) over T *if* $|Q_{\text{LR}}(y)| = |Q_{\text{LR}}(yz)|$ *for all* $yz \in T$. *It is* RL-generic *if* $|Q_{\text{RL}}(zy)| = |Q_{\text{RL}}(y)|$ *for all* $zy \in T$. *It is* generic *if it is both* LR- *and* RL-*generic.*

Lemma 2. *Every* $\emptyset \neq T \subseteq \Sigma^*$ *admits* LR- *and* RL-*generic strings. Also, if* y_{L} *is* LR-*generic and* y_{R} *is* RL-*generic, then every* $y_{\text{L}}xy_{\text{R}} \in T$ *is generic over* T.

Alternatively, genericity can be characterized via $\alpha_{y,z}$ and $\beta_{z,y}$, as follows.

Lemma 3. *Let $y \in T \subseteq \Sigma^*$. Then y is* LR*-generic over T iff $\alpha_{y,z}$ is total and bijective from $Q_{\mathrm{LR}}(y)$ to $Q_{\mathrm{LR}}(yz)$ for all $yz \in T$. Similarly, y is* RL*-generic over T iff $\beta_{z,y}$ is total and bijective from $Q_{\mathrm{RL}}(y)$ to $Q_{\mathrm{RL}}(zy)$ for all $zy \in T$.*

Proof. We focus on the first equivalence (the second one follows symmetrically) and on the 'only if' direction —the 'if' direction is immediate, since the existence of any total bijection from $Q_{\mathrm{LR}}(y)$ to $Q_{\mathrm{LR}}(yz)$ implies $|Q_{\mathrm{LR}}(y)| = |Q_{\mathrm{LR}}(yz)|$.

Let y be LR-generic over T and pick $yz \in T$. We know $\alpha_{y,z}$ partially maps $Q_{\mathrm{LR}}(y)$ to Q (by definition) and covers $Q_{\mathrm{LR}}(yz)$ (Fact 2a). Namely, each $r \in Q_{\mathrm{LR}}(yz)$ has a distinct $q \in Q_{\mathrm{LR}}(y)$ with $\alpha_{y,z}(q) = r$. So, if there were $q \in Q_{\mathrm{LR}}(y)$ with $\alpha_{y,z}(q)$ undefined or outside $Q_{\mathrm{LR}}(yz)$ or equal to $\alpha_{y,z}(q')$ for another $q' \in Q_{\mathrm{LR}}(y)$, we would have $|Q_{\mathrm{LR}}(y)| > |Q_{\mathrm{LR}}(yz)|$, contrary to y being generic. Hence, $\alpha_{y,z}(q)$ is defined and in $Q_{\mathrm{LR}}(yz)$ and distinct, for all $q \in Q_{\mathrm{LR}}(y)$. Namely, $\alpha_{y,z}$ is a total injection from $Q_{\mathrm{LR}}(y)$ to $Q_{\mathrm{LR}}(yz)$. By Fact 2a, it is also a surjection. □

3.2 Blocks

Fix $\emptyset \neq T \subseteq \Sigma^*$, fix a generic ϑ over T, and let $A := Q_{\mathrm{LR}}(\vartheta)$ and $B := Q_{\mathrm{RL}}(\vartheta)$.

Every string of the form $\vartheta x \vartheta$ is a *block* (*on* ϑ), and x is its *infix*. We say the pair $(\alpha_x, \beta_x) := (\alpha_{\vartheta,x\vartheta}, \beta_{\vartheta x,\vartheta})$ are the *inner behavior* of M on the block.

Recall that $\alpha_x : A \rightharpoonup Q$ and $\beta_x : B \rightharpoonup Q$. In the special case where each function is the identity, the prefix ϑx and the suffix $x\vartheta$ are 'invisible' to M.

Lemma 4. *Suppose $(\alpha_x, \beta_x) = (\mathrm{id}_A, \mathrm{id}_B)$. Pick any u,v and let $c_0, c_1, \ldots$ and $d_0, d_1, \ldots$ be the decompositions of $\mathrm{COMP}(u\vartheta v)$ and $\mathrm{COMP}(u\vartheta x\vartheta v)$ by ϑ and $\vartheta x\vartheta$, respectively. Then c_i parallels d_i for all even i, and resembles d_i for all odd i. Thus, M behaves (accepts, rejects, or loops) identically on $u\vartheta v$ and $u\vartheta x\vartheta v$.*

In blocks of the form $\vartheta(x\vartheta y)\vartheta$, where ϑ is an infix of the infix itself, the inner behavior of M depends on its inner behaviors on the sub-blocks $\vartheta x\vartheta$ and $\vartheta y\vartheta$.

Fact 3. *Let $z = x\vartheta y$. Then $\alpha_x \circ \alpha_y \leq \alpha_z$ and $\beta_y \circ \beta_x \leq \beta_z$. In addition, if α_z is total and injective, then so is α_x; if β_z is total and injective, then so is β_y.*

Proof. To prove $\alpha_x \circ \alpha_y \leq \alpha_z$, let $p \in A$ and assume $(\alpha_x \circ \alpha_y)(p)$ is defined and equal to some $r \in Q$. Then $\alpha_x(p)$ is defined and equal to some $q \in Q$, and $\alpha_y(q)$ is defined and equal to r. By $\alpha_x(p) = q$, we know $c_x := \mathrm{COMP}_{p,|\vartheta|+1}(\vartheta x\vartheta)$ hits right into q. (Fig. 1e.) By $\alpha_y(q) = r$, we also know $c_y := \mathrm{COMP}_{q,|\vartheta|+1}(\vartheta y\vartheta)$ hits right into r. Now, concatenating c_x, c_y gives exactly $c_z := \mathrm{COMP}_{p,|\vartheta|+1}(\vartheta x\vartheta y\vartheta)$. Hence c_z hits right into r. Therefore $\alpha_z(p)$ is defined and equal to $(\alpha_x \circ \alpha_y)(p)$.

Now suppose α_z is total and injective. *If α_x is not total*, then $\alpha_x(p)$ is undefined for some $p \in A$, namely $c_x := \mathrm{COMP}_{p,|\vartheta|+1}(\vartheta x\vartheta)$ hits left or loops. But c_x is a prefix of $c_z := \mathrm{COMP}_{p,|\vartheta|+1}(\vartheta x\vartheta y\vartheta)$, so c_z also hits left or loops. Hence $\alpha_z(p)$ is undefined, and α_z is not total—contradiction. *If α_x is not injective*, then $\alpha_x(p) = \alpha_x(p')$ for two distinct $p, p' \in A$, namely $c_x := \mathrm{COMP}_{p,|\vartheta|+1}(\vartheta x\vartheta)$ and $c'_x := \mathrm{COMP}_{p',|\vartheta|+1}(\vartheta x\vartheta)$ hit right into the same state. But c_x and c'_x are prefixes of $c_z := \mathrm{COMP}_{p,|\vartheta|+1}(\vartheta x\vartheta y\vartheta)$ and $c'_z := \mathrm{COMP}_{p',|\vartheta|+1}(\vartheta x\vartheta y\vartheta)$, so c_z and c'_z continue identically after the $\vartheta x\vartheta$-$y\vartheta$ boundary, hitting right into the same state. Hence $\alpha_z(p) = \alpha_z(p')$, and α_z is not injective—contradiction. □

We will need a variant of Fact 3 for blocks of the form $\vartheta(x\vartheta x\vartheta\cdots x\vartheta x)\vartheta$, where the infix is multiple ϑ-separated copies of x. Let $x^{(k)} := x(\vartheta x)^{k-1}$ for $k \geq 1$. Note that $\vartheta x^{(k)}\vartheta = \vartheta(x\vartheta)^k = (\vartheta x)^k\vartheta$ and $(x^{(k)})^{(l)} = x^{(lk)}$ for all k, l.

Fact 4. *Let $k \geq 1$. Then $(\alpha_x)^k \leq \alpha_{x^{(k)}}$ and $(\beta_x)^k \leq \beta_{x^{(k)}}$. In addition, if $\alpha_{x^{(k)}}$ is total and injective, then so is α_x; if $\beta_{x^{(k)}}$ is total and injective, then so is β_x.*

Proof. We prove $(\alpha_x)^k \leq \alpha_{x^{(k)}}$ inductively. Case $k = 1$ is trivial. For the inductive step, assume $(\alpha_x)^k \leq \alpha_{x^{(k)}}$. Then $(\alpha_x)^{k+1} = \alpha_x \circ (\alpha_x)^k \leq \alpha_x \circ \alpha_{x^{(k)}}$ (Fact 1) and $\alpha_x \circ \alpha_{x^{(k)}} \leq \alpha_{x^{(k+1)}}$ (Fact 3 for $z = x\vartheta(x^{(k)}) = x\vartheta x(\vartheta x)^{k-1} = x(\vartheta x)^k = x^{(k+1)}$). So, $(\alpha_x)^{k+1} \leq \alpha_{x^{(k+1)}}$ (by transitivity of $\leq$), and we are done. The additional claim follows from that of Fact 3 when $z = x\vartheta(x^{(k-1)}) = x^{(k)}$. □

If any infix $x^{(k)}$ causes the inner behavior of M to just permute the outcomes of ϑ, then longer infixes force the behavior into the special case of Lemma 4.

Fact 5. *If $(\alpha_{x^{(k)}}, \beta_{x^{(k)}})$ permute (A, B), then $(\alpha_{x^{(tlk)}}, \beta_{x^{(tlk)}}) = (\mathrm{id}_A, \mathrm{id}_B)$ for some $l \geq 1$ and all $t \geq 1$.*

Proof. Let $z := x^{(k)}$ and suppose α_z and β_z are permutations of A and B. Pick $l \geq 1$ so that both permutations become identity after l iterations: $(\alpha_z)^l = \mathrm{id}_A$ and $(\beta_z)^l = \mathrm{id}_B$. Then $(\alpha_z)^l \leq \alpha_{z^{(l)}}$ (Fact 4), where $z^{(l)} = (x^{(k)})^{(l)} = x^{(lk)}$; i.e., $\mathrm{id}_A \leq \alpha_{x^{(lk)}}$, so $\alpha_{x^{(lk)}} = \mathrm{id}_A$ (Fact 1). Similarly, $\beta_{x^{(lk)}} = \mathrm{id}_B$. Now, let $t \geq 1$. By Fact 4, $(\alpha_{x^{(lk)}})^t \leq \alpha_{(x^{(lk)})^{(t)}}$. By $(\alpha_{x^{(lk)}})^t = (\mathrm{id}_A)^t = \mathrm{id}_A$ and $(x^{(lk)})^{(t)} = x^{(tlk)}$, we know $\mathrm{id}_A \leq \alpha_{x^{(tlk)}}$, so $\alpha_{x^{(tlk)}} = \mathrm{id}_A$ (Fact 1). Similarly, $\beta_{x^{(tlk)}} = \mathrm{id}_B$. □

We will now state a condition that forces the number of reversals to become more than sublinear. We say (A, B) *use reversals on x* if some $\mathrm{COMP}_{p,|\vartheta|+1}(\vartheta x\vartheta)$ for $p \in A$ or some $\mathrm{COMP}_{p,|\vartheta x|}(\vartheta x\vartheta)$ for $p \in B$ contains at least one reversal.

Lemma 5. *If (A, B) use reversals on x and (α_x, β_x) permute (A, B), then it cannot be $r_M(n) = o(n)$.*

Proof. Since (A, B) use reversals, there is a $d := \mathrm{COMP}_{q,|\vartheta|+1}(\vartheta x\vartheta)$ with $q \in A$ (or a $\mathrm{COMP}_{q,|\vartheta x|}(\vartheta x\vartheta)$ with $q \in B$, and we work similarly) containing ≥ 1 reversal (Fig. 1f); in fact, d contains ≥ 1 *forward* reversal (because d hits right, since α_x permutes $A \implies \alpha_x(q)$ is defined). Since (α_x, β_x) permute (A, B), there exists $l \geq 1$ such that $(\alpha_{x^{(tl)}}, \beta_{x^{(tl)}}) = (\mathrm{id}_A, \mathrm{id}_B)$ for all $t \geq 1$ (by Fact 5 for $k = 1$).

Let $z := x^{(l)}$. Then $z^{(t)} = x^{(tl)}$ and thus $(\alpha_{z^{(t)}}, \beta_{z^{(t)}}) = (\mathrm{id}_A, \mathrm{id}_B)$, for all t. Using this, we show that each $d_t := \mathrm{COMP}_{q,|\vartheta|+1}(\vartheta z^{(t)}\vartheta)$ reverses a lot (Fig. 1g).

Claim. For every $t \geq 1$, computation d_t contains $\geq t$ forward reversals.

Proof. By induction. For $t = 1$, $d_1 = \mathrm{COMP}_{q,|\vartheta|+1}(\vartheta z^{(1)}\vartheta)$. By $z^{(1)} = z = x^{(l)}$ and $l \geq 1$, we know $\vartheta z^{(1)}\vartheta$ has $\vartheta x\vartheta$ as prefix, hence d_1 has d as prefix, and thus contains ≥ 1 forward reversals. For $t > 0$, $d_t = \mathrm{COMP}_{q,|\vartheta|+1}(\vartheta z^{(t)}\vartheta)$. Since $\vartheta z^{(t)}\vartheta = \vartheta z^{(t-1)}\vartheta z\vartheta$, the prefix of d_t up to the $\vartheta z^{(t-1)}\vartheta$-$z\vartheta$ boundary is d_{t-1}, the state after crossing this boundary is $\alpha_{z^{(t-1)}}(q) = \mathrm{id}_A(q) = q$, and thus the remaining suffix $\mathrm{COMP}_{q,|\vartheta z^{(t-1)}\vartheta|+1}(\vartheta z^{(t-1)}\vartheta z\vartheta)$ parallels d_1. Hence, d_t contains the $\geq t-1$ forward reversals of d_{t-1} plus the ≥ 1 of d_1, for a total of $\geq t$. ⊡

Since $q \in A = Q_{\text{LR}}(\vartheta)$, there exist p,u,v such that $c := \text{LCOMP}_p(\vartheta)$ appears in $\hat{c} := \text{COMP}(u\vartheta v)$ and hits right into q (Fig. 1h). Consider the family of inputs $w_t := u\vartheta z^{(t)}\vartheta v$ for $t \geq 1$, and the respective computations $\hat{c}_t := \text{COMP}(w_t)$ (Fig. 1i). By Lemma 4 and $(\alpha_{z^{(t)}}, \beta_{z^{(t)}}) = (\text{id}_A, \text{id}_B)$, we know c resembles a segment c_t in the decomposition of $\hat{c}_t$ by $\vartheta z^{(t)}\vartheta$. So, c_t is a L-computation on $\vartheta z^{(t)}\vartheta$ from p. Since ϑ is a prefix of $\vartheta z^{(t)}\vartheta$, the prefix of c_t up to the ϑ-$z^{(t)}\vartheta$ boundary is c, the state after crossing the boundary is q, and the suffix $\text{COMP}_{q,|\vartheta|+1}(\vartheta z^{(t)}\vartheta)$ from then on parallels d_t. So, $\hat{c}_t$ also contains $\geq t$ forward reversals, and thus $\geq 2t$ reversals overall ($\hat{c}_t$ is full, so each forward reversal follows a backward one).

Now, each $\hat{c}_t$ works on input length $n_t := |w_t| = |u\vartheta x^{(tl)}\vartheta v| = |u\vartheta(x\vartheta)^{tl}v| = l|x\vartheta| \cdot t + |u\vartheta v|$. Thus $r_M(n_t) \geq r(\hat{c}_t) \geq 2t = (2/l|x\vartheta|) \cdot n_t - 2|u\vartheta v|/l|x\vartheta|$. So, $r_M(n)$ exceeds a linear function inifinitely often. Hence, $r_M(n) \neq o(n)$. □

Now fix $\tilde{T} \subseteq \overline{T}$. With respect to the problem $(T, \tilde{T})$, an infix x is *positive*, *negative*, or *neutral* if $\vartheta x\vartheta$ is in T, in $\tilde{T}$, or in neither. We will encounter cases which meet the promise that *either some $x^{(k)}$ are positive or all $x^{(k)}$ are negative*. We then say that ϑ,x *respect* $(T, \tilde{T})$; and that they *select* T (resp., $\tilde{T}$), if the promise is met by its left (resp., right) disjunct. If in addition M solves $(T, \tilde{T})$, then we can tell which disjunct is selected using a 'local' criterion for M on $\vartheta x\vartheta$. The next fact assembles this criterion; the next lemma states it more compactly.

Fact 6. *If positive $x^{(k)}$ exist, then (α_x, β_x) permute (A, B). Almost conversely, if M solves $(T, \tilde{T})$ and (α_x, β_x) permute (A, B), then non-negative $x^{(k)}$ exist.*

Proof. [⇒] Suppose $z := x^{(k)}$ is positive for some $k \geq 1$. We shall prove that $\alpha_x : A \rightharpoonup Q$ is a permutation of A (the claim for β_x follows similarly). For this, it is enough to prove two Claims: (1) α_x is total and injective, and (2) $\alpha_x[A] \subseteq A$.

Since z is positive, namely $\vartheta z\vartheta = \vartheta(x\vartheta)^k \in T$, we know $\alpha_z = \alpha_{\vartheta,(x\vartheta)^k}$ is a total bijection from $A = Q_{\text{LR}}(\vartheta)$ to $A' := Q_{\text{LR}}(\vartheta z\vartheta)$ (Lemma 3). But $A' \subseteq A$ (Fact 2a, since $\vartheta z\vartheta$ ends in ϑ) and $|A'| = |A|$ (since α_z is bijective), so $A' = A$. Thus, $\alpha_z = \alpha_{x^{(k)}}$ permutes A. By a symmetric argument, $\beta_z = \beta_{x^{(k)}}$ permutes B.

Since $\alpha_{x^{(k)}}$ is total and injective, Claim 1 is true (Fact 4). For Claim 2, let $r \in \alpha_x[A]$. Then there is $q \in A = Q_{\text{LR}}(\vartheta)$ with $\alpha_x(q) = r$. I.e., there exist p,q and u,v such that $c := \text{LCOMP}_p(\vartheta)$ appears in $\hat{c} := \text{COMP}(u\vartheta v)$ and hits right into q (Fig. 1h), and $d := \text{COMP}_{q,|\vartheta|+1}(\vartheta x\vartheta)$ hits right into r (Fig. 1f). Note that c is an odd-indexed segment in the decomposition of $\hat{c}$ by ϑ. Now pick any $t \geq 1$ with $(\alpha_{z^{(t)}}, \beta_{z^{(t)}}) = (\alpha_{x^{(tk)}}, \beta_{x^{(tk)}}) = (\text{id}_A, \text{id}_B)$ (Fact 5). Lemma 4 says c resembles an odd-indexed segment c_t in the decomposition of $\hat{c}_t := \text{COMP}(u\vartheta z^{(t)}\vartheta v)$ by $\vartheta z^{(t)}\vartheta$ (Fig. 1j). So, c_t is also a L-computation from p, on $\vartheta z^{(t)}\vartheta$. Since $\vartheta x\vartheta$ is a prefix of $\vartheta z^{(t)}\vartheta$, the prefix of c_t up to the first crossing of the right boundary of $\vartheta x\vartheta$ is c followed by a parallel of d. In particular, if $\tilde{q}$ is the state in d after the last crossing of the ϑx-ϑ boundary, then $\tilde{d} := \text{LCOMP}_{\tilde{q}}(\vartheta)$ hits right into r and appears in $\hat{c}_t = \text{COMP}((u\vartheta x)\vartheta(x^{(tk-1)}\vartheta v))$. Hence, $r \in Q_{\text{LR}}(\vartheta) = A$.

[⇐] Suppose M solves $(T, \tilde{T})$ and $(\alpha_x, \beta_x) = (\alpha_{x^{(1)}}, \beta_{x^{(1)}})$ permute (A, B). Pick any $t \geq 1$ with $(\alpha_{x^{(t\cdot 1)}}, \beta_{x^{(t\cdot 1)}}) = (\text{id}_A, \text{id}_B)$ (Fact 5). Pick $k = t \cdot 1$. Then M behaves identically on ϑ and $\vartheta x^{(k)}\vartheta$ (Lemma 4 with empty u,v). Since it accepts $\vartheta \in T$, it also accepts $\vartheta x^{(k)}\vartheta$, thus $\vartheta x^{(k)}\vartheta \notin \tilde{T}$. So, $x^{(k)}$ is positive or neutral. □

Lemma 6. *Suppose M solves $(T,\tilde{T})$ and ϑ,x respect $(T,\tilde{T})$. Then ϑ,x select T iff each outcome of ϑ is hit exactly once by the respective half of the inner behavior:*

$$\big(\forall r \in A\big)\big(|\alpha_x^{-1}(r)| = 1\big) \quad \& \quad \big(\forall r \in B\big)\big(|\beta_x^{-1}(r)| = 1\big)\,. \tag{2}$$

Proof. If ϑ,x select T, then positive $x^{(k)}$ exist, so α_x permutes A (Fact 6) and thus hits every $r \in A$ exactly once; similarly for β_x,B. Conversely, if $\alpha_x : A \rightharpoonup Q$ hits every $r \in A$, then it is total and injective and stays in A (or else its values are not enough to cover A), hence it bijects A into A, i.e., permutes it; similarly for β_x,B. So, nonnegative $x^{(k)}$ exist (Fact 6). So ϑ,x do not select $\tilde{T}$, but T. □

If M uses sublinearly many reversals, then we can simplify (2) by replacing $\alpha_x^{-1}(r), \beta_x^{-1}(r)$ by two simpler sets $\alpha_x^*(r), \beta_x^*(r)$, which we now introduce. Recall that $\alpha_x^{-1}(r)$ is all $p \in A$ for which $\textsc{comp}_{p,|\vartheta|+1}(\vartheta x \vartheta)$ hits right into r. Of course, each p may reach r after arbitrary meanders inside $\vartheta x \vartheta$. Now suppose we demand computations that stay inside $x\vartheta$ and cross the x-ϑ boundary only once; then $\alpha_x^*(r)$ is all $p \in A$ that reach r via such 'simple computations' (Fig. 1k):

$$\alpha_x^*(r) = \alpha_{\vartheta,x\vartheta}^*(r) := \{p \in A \mid (\exists q \in Q)(\textsc{lcomp}_p(x) \text{ hits right into } q \\ \& \ \textsc{lcomp}_q(\vartheta) \text{ hits right into } r)\}\,. \tag{3}$$

Symmetrically, $\beta_x^*(r) = \beta_{\vartheta x,\vartheta}^*(r)$ is all $p \in B$ for which $\textsc{comp}_{p,|\vartheta x|}(\vartheta x \vartheta)$ hits left into r having crossed the ϑx-ϑ and ϑ-$x\vartheta$ boundaries 0 and 1 times respectively.

The simplification of (2) is proved in the next lemma. Before that, the next fact studies the new sets. The boolean functions $\delta_{\text{LR}}(p,x,q)$ and $\delta_{\text{RL}}(q,x,p)$ are 1 iff $\textsc{lcomp}_p(x)$ hits right into q and iff $\textsc{rcomp}_p(x)$ hits left into q, respectively.

Fact 7. *For all $r \in Q$: $\alpha_x^*(r) \subseteq \alpha_x^{-1}(r)$ and $\beta_x^*(r) \subseteq \beta_x^{-1}(r)$. Moreover:*

$$|\alpha_x^*(r)| = \sum_{\substack{p \in A \ \&\ \textsc{lcomp}_q(\vartheta) \\ \text{hits right into } r}} \delta_{\text{LR}}(p,x,q) \qquad |\beta_x^*(r)| = \sum_{\substack{p \in B \ \&\ \textsc{rcomp}_q(\vartheta) \\ \text{hits left into } r}} \delta_{\text{RL}}(q,x,p)\,. \tag{4}$$

Proof. The inclusions are easy. For the left equality, consider any $p \in A$ and the inner sum $S_p := \sum_q \delta_{\text{LR}}(p,x,q)$ over all q for which $\textsc{lcomp}_q(\vartheta)$ hits right into r (Fig. 1l). Since M is deterministic, $S_p \le 1$. And $S_p = 1$ iff one of these q is the witness required in (3); i.e., $S_p = 1 \iff p \in \alpha_x^*(r)$. So, the number $\sum_{p \in A} S_p$ of $p \in A$ for which $S_p = 1$, is the size of $\alpha_x^*(r)$. Similarly for the other equality. □

Lemma 7. *Suppose M solves $(T,\tilde{T})$ with $r_M(n) = o(n)$, and ϑ,x respect $(T,\tilde{T})$. Then ϑ,x select T iff each outcome of ϑ is hit by exactly one 'simple computation':*

$$\big(\forall r \in A\big)\big(|\alpha_x^*(r)| = 1\big) \quad \& \quad \big(\forall r \in B\big)\big(|\beta_x^*(r)| = 1\big)\,. \tag{5}$$

Proof. Suppose ϑ,x select T. Then (α_x, β_x) permute (A,B) (Fact 6), so (A,B) do not use reversals (Lemma 5 and $r_M(n) = o(n)$). Now pick any $r \in A$. Then $|\alpha_x^*(r)| \le 1$, because $\alpha_x^*(r) \subseteq \alpha_x^{-1}(r)$ (Fact 7) and $|\alpha_x^{-1}(r)| = 1$ (Lemma 6). And $|\alpha_x^*(r)| \ge 1$, because the r-hitting $\textsc{comp}_{p,|\vartheta|+1}(\vartheta x \vartheta)$ of the unique $p \in \alpha_x^{-1}(r)$ uses no reversals (because (A,B) do not use reversals), thus $p \in \alpha_x^*(r)$. Overall, $|\alpha_x^*(r)| = 1$. Similarly for β_x^*,B. Conversely, if (5) is true, then (2) is true (since $\alpha_x^*(r) \subseteq \alpha_x^{-1}(r)$ and $\beta_x^*(r) \subseteq \beta_x^{-1}(r)$), and thus ϑ,x select T (Lemma 6). □

4 The Proof

Fix $h \geq 1$. Suppose $\Sigma = \Sigma_h$ and M solves OWL_h with $r_M(n) = o(n)$ reversals. We will prove that M needs exponentially many states, namely $|Q| = \Omega(2^h)$.

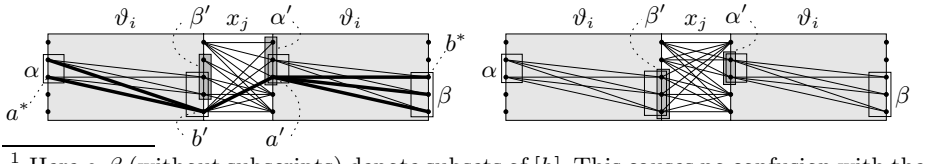
4.1 The Hard Instances

We focus on a family of hard instances of OWL_h similar to that of [6, §3.2]. This is all blocks $\vartheta x \vartheta$ where ϑ and x are drawn from two families $(\vartheta_i)_{i\in\mathcal{I}}$ and $(x_i)_{i\in\mathcal{I}}$ of generic and single-symbol strings, respectively. Here, i ranges over all pairs of non-empty subsets of $[h]$, namely $\mathcal{I} := \{(\alpha,\beta) \mid \emptyset \neq \alpha,\beta \subseteq [h]\}$.[1] These are totally ordered by the rule $(\alpha',\beta') < (\alpha,\beta) \Leftrightarrow_{\text{def}} \langle\alpha'\rangle\langle\beta'\rangle <_{\text{b}} \langle\alpha\rangle\langle\beta\rangle$, where $\langle\cdot\rangle$ is the natural h-bit encoding of subsets of $[h]$ and $<_{\text{b}}$ is the natural order on $2h$-bit positive integers. For each $i = (\alpha,\beta) \in \mathcal{I}$, the string ϑ_i is any fixed generic string over $T_i := \{z \in \Sigma^* \mid z \text{ has connectivity } \alpha \times \beta\}$, and $x_i := \overline{\beta \times \alpha}$ is the 1-long string of all arrows not in $\beta \times \alpha$. We also let $T_\emptyset := \{z \in \Sigma^* \mid z \text{ is dead}\}$.

We picture these blocks on a $|\mathcal{I}| \times |\mathcal{I}|$ matrix. Cell (i,j) hosts block $\vartheta_i x_j \vartheta_i$ and copies of the objects associated with it in Lemma 7: the sets $A_i := Q_{\text{LR}}(\vartheta_i)$, $B_i := Q_{\text{RL}}(\vartheta_i)$ and the functions $\alpha^*_{i,j} := \alpha^*_{\vartheta_i x_j \vartheta_i}$, $\beta^*_{i,j} := \beta^*_{\vartheta_i x_j, \vartheta_i}$. Crucially, the assumptions of Lemma 7 are satisfied in all cells, and its conclusions follow a simple pattern on and below the diagonal (i.e., when $i \geq j$).

Fact 8. *For all $i,j \in \mathcal{I}$, the assumptions of Lemma 7 are satisfied by M,ϑ_i,x_j for $(T_i, T_\emptyset)$. Furthermore, if $i > j$ then ϑ_i,x_j select T_i; if $i = j$ then ϑ_i,x_j select $T_\emptyset$.*

Proof. Fix any $i = (\alpha,\beta)$ and $j = (\alpha',\beta')$. We first check the assumptions of the lemma. Easily, M solves $(T_i, T_\emptyset)$ (since all of T_i is live and all of $T_\emptyset$ is dead) with $r_M(n) = o(n)$ (by assumption), and ϑ_i is generic for M over T_i (by selection). To show that ϑ_i,x_j respect $(T_i, T_\emptyset)$, we take cases. *If $\vartheta_i x_j \vartheta_i$ is dead*, then all $\vartheta_i (x_j \vartheta_i)^k$ for $k \geq 1$ are dead (since all extensions of a dead string are dead), namely all $(x_j)^{(k)}$ are negative. *If $\vartheta_i x_j \vartheta_i$ is live*, then some path $a^* \rightsquigarrow b^*$ for a^*,$b^* \in [h]$ connects its outer columns (cf. next figure, left side). If b',a' are the visited nodes on the columns of x_j, then the path has the form $a^* \rightsquigarrow b' \to a' \rightsquigarrow b^*$ and ϑ_i has paths $a^* \rightsquigarrow b'$ and $a' \rightsquigarrow b^*$. Hence $(a^*, b'), (a', b^*) \in \xi$, for $\xi = \alpha \times \beta$ the connectivity of ϑ_i. Thus, $b' \in \beta$ and $a' \in \alpha$. Now, for any a,$b \in [h]$, consider the ath leftmost and bth rightmost nodes of $\vartheta_i x_j \vartheta_i$. If $a \notin \alpha \ \vee \ b \notin \beta$, then the two nodes do not connect, since neither can 'see through' ϑ_i; but if $a \in \alpha$ & $b \in \beta$, then $(a, b'), (a', b) \in \xi$, so the two nodes connect via a path $a \rightsquigarrow b' \to a' \rightsquigarrow b$. Hence, $\vartheta_i x_j \vartheta_i$ has connectivity ξ, namely $\vartheta_i x_j \vartheta_i \in T_i$, and $(x_j)^{(1)}$ is positive. Overall, ϑ_i,x_j respect $(T_i, T_\emptyset)$. In particular, ϑ_i,x_j *select T_i iff $\vartheta_i x_j \vartheta_i$ is live.*



[1] Here α,β (without subscripts) denote subsets of $[h]$. This causes no confusion with the names (with subscripts) for M's inner behavior, and preserves notational symmetry.

If $i > j$ (cf. left side), then $\langle\alpha'\rangle\langle\beta'\rangle <_{\mathrm{b}} \langle\alpha\rangle\langle\beta\rangle$. Thus $\alpha' \not\supseteq \alpha \;\vee\; \beta' \not\supseteq \beta$ (otherwise $\alpha' \supseteq \alpha$ & $\beta' \supseteq \beta$, thus the 1's of $\langle\alpha'\rangle$ and $\langle\beta'\rangle$ cover all 1's of $\langle\alpha\rangle$ and $\langle\beta\rangle$, hence $\langle\alpha'\rangle\langle\beta'\rangle \geq_{\mathrm{b}} \langle\alpha\rangle\langle\beta\rangle$, a contradiction). Suppose $\beta' \not\supseteq \beta$ (if $\alpha' \not\supseteq \alpha$, apply a similar argument). Pick any $a^* \in \alpha$, $b' \in \beta \setminus \beta'$, $a' \in \alpha$, and $b^* \in \beta$. Then $(a^*, b') \in \xi$ and $(b', a') \in \overline{\beta' \times \alpha'}$ and $(a', b^*) \in \xi$, therefore $\vartheta_i x_j \vartheta_i$ contains the path $a^* \rightsquigarrow b' \to a' \rightsquigarrow b^*$, and is live. Thus, ϑ_i,x_j select $\underline{T_i}$.

If $i = j$ (cf. right side), then x_j has connectivity $\xi' = \overline{\beta \times \alpha}$. If ϑ_i,x_j do not select $T_\emptyset$, then they select T_i, so $\vartheta_i x_j \vartheta_i$ is live. Pick any witnessing path, say of the form $a^* \rightsquigarrow b' \to a' \rightsquigarrow b^*$. Then $(a^*, b') \in \xi$ and $(b', a') \in \xi'$ and $(a', b^*) \in \xi$. Therefore $b' \in \beta$ and $(b', a') \in \overline{\beta \times \alpha}$ and $a' \in \alpha$, a contradiction. □

4.2 The Bound

Now consider the following two families of experiments.

First, fix ϑ_i and $r \in Q$, let x_j range over all possibilities, and observe how the sizes of $\alpha^*_{i,j}(r)$ and $\beta^*_{i,j}(r)$ vary with x_j. The result is two $1 \times |\mathcal{I}|$ vectors, $\mathrm{a}_{i,r} := (|\alpha^*_{i,j}(r)|)_{j\in\mathcal{I}}$ and $\mathrm{b}_{i,r} := (|\beta^*_{i,j}(r)|)_{j\in\mathcal{I}}$. Repeat for all ϑ_i and r, to obtain the two sets of vectors $\mathcal{A} := \{\mathrm{a}_{i,r} \mid i \in \mathcal{I}, r \in Q\}$ and $\mathcal{B} := \{\mathrm{b}_{i,r} \mid i \in \mathcal{I}, r \in Q\}$.

Second, fix p,$q \in Q$, let x_j range over all possibilities, and observe how the bits $\delta_{\mathrm{LR}}(p, x_j, q)$ and $\delta_{\mathrm{RL}}(q, x_j, p)$ vary. The result is two $1 \times |\mathcal{I}|$ binary vectors, $\mathrm{u}_{p,q} := (\delta_{\mathrm{LR}}(p, x_j, q))_{j\in\mathcal{I}}$ and $\mathrm{v}_{q,p} := (\delta_{\mathrm{RL}}(q, x_j, p))_{j\in\mathcal{I}}$. Repeat for all p and q, to obtain the two sets of vectors $\mathcal{U} := \{\mathrm{u}_{p,q} \mid p, q \in Q\}$ and $\mathcal{V} := \{\mathrm{v}_{q,p} \mid p, q \in Q\}$.

Fact 9a. *Every vector in $\mathcal{A} \cup \mathcal{B}$ is a linear combination of vectors from $\mathcal{U} \cup \mathcal{V}$.*

Proof. Fix $i \in \mathcal{I}$ and $r \in Q$. The left equality in Fact 7 implies that for all $j \in \mathcal{I}$:

$$\mathrm{a}_{i,r}(j) = |\alpha^*_{i,j}(r)| = |\alpha^*_{\vartheta_i,x_j\vartheta_i}(r)| = \sum_{\substack{p\in A_i \text{ \& } \textsc{lcomp}_q(\vartheta_i) \\ \text{hits right into } r}} \delta_{\mathrm{LR}}(p, x_j, q) = \sum_{\substack{p\in A_i \text{ \& } \textsc{lcomp}_q(\vartheta_i) \\ \text{hits right into } r}} \mathrm{u}_{p,q}(j)$$

and thus $\mathrm{a}_{i,r} = \sum \mathrm{u}_{p,q}$ for the specific ranges of p,q. Similarly, $\mathrm{b}_{i,r} = \sum \mathrm{v}_{q,p}$ if the sum ranges over all $p \in B_i$ and all q for which $\textsc{rcomp}_q(\vartheta_i)$ hits left into r. □

Fact 9b. *The set $\mathcal{A} \cup \mathcal{B}$ contains $|\mathcal{I}| - 1$ linearly independent vectors.*

Proof. We will find in $\mathcal{A} \cup \mathcal{B}$ a vector family $(\mathrm{c}_i)_{i\in\mathcal{I}}$ such that $i > j \implies \mathrm{c}_i(j) = 1$ and $i = j \implies \mathrm{c}_i(j) = 0$, for all i,$j \in \mathcal{I}$. This will be enough. Because then the numbers $\mathrm{c}_i(j)$ form a $|\mathcal{I}| \times |\mathcal{I}|$ matrix with 0s on the diagonal and 1s below it, which has rank $|\mathcal{I}| - 1$ (easily), and thus $|\mathcal{I}| - 1$ of the c_i must be independent.

Pick $i \in \mathcal{I}$. Since ϑ_i, x_i select $T_\emptyset$ (Fact 8), there exist $r \in A_i$ or $r \in B_i$ which are *not* hit by exactly 1 simple computation (Lemma 7), respectively $|\alpha^*_{i,i}(r)| \neq 1$ or $|\beta^*_{i,i}(r)| \neq 1$. One of these r must, in fact, be hit by 0 simple computations (otherwise, each r is hit by ≥ 1 value of $\alpha_{i,i}$ or $\beta_{i,i}$ and at least one is hit by ≥ 2 values, for an absurd total of $\geq |A_i| + |B_i| + 1$ values of $\alpha_{i,i}$ and $\beta_{i,i}$). If r_i is this unhit state, then $r_i \in A_i$ & $|\alpha^*_{i,i}(r_i)| = 0$ or $r_i \in B_i$ & $|\beta^*_{i,i}(r_i)| = 0$. In the former case, we let $\mathrm{c}_i := \mathrm{a}_{i,r_i}$; in the latter case, we let $\mathrm{c}_i := \mathrm{b}_{i,r_i}$.

By this definition, clearly $\mathrm{c}_i(i) = 0$. Moreover, if $i > j$ then ϑ_i,x_j select T_i (Fact 8) and thus each $r \in A_i$ and $r \in B_i$ is hit by exactly 1 simple computation (Lemma 7). Hence, so is r_i. Therefore, depending on the case in c_i's definition, either $\mathrm{c}_i(j) = \mathrm{a}_{i,r_i}(j) = |\alpha^*_{i,j}(r_i)| = 1$ or $\mathrm{c}_i(j) = \mathrm{b}_{i,r_i}(j) = |\beta^*_{i,j}(r_i)| = 1$. □

So, $\mathcal{U} \cup \mathcal{V}$ span a space of dimension $\geq |\mathcal{I}| - 1$. Clearly then $|\mathcal{U} \cup \mathcal{V}| \geq |\mathcal{I}| - 1$, therefore $2|Q|^2 \geq (2^h - 1)^2 - 1$. Hence $|Q| = \Omega(2^h)$, and the proof is complete.

5 Conclusion

We confirmed the Sakoda-Sipser conjecture in the special case of 2FAs performing $o(n)$ reversals, by proving that OWL needs exponentially large 2DFAs of this kind.

The large alphabet of OWL_h is not a problem, as binary witnesses exist, too: Just encode the symbols of Σ_h into h^2-bit strings. Then 2DFAs still need $\Omega(2^h)$ states (same proof, as all reasoning is on cell boundaries), while 2NFAs need only $O(h^2)$. So, our title is valid even if 'small 2FAs' means '2FAs of small description'.

Theorem 1 says that all 2DFAs for OWL_h satisfy $r_M(n) \neq o(n) \vee |Q| = \Omega(2^h)$, but the stronger condition $r_M(n) = \Omega(n) \vee |Q| \geq 2^h$ is probably also true. Another possible direction for further work is to continue with Research Problem 4 of [4] and fully analyze the trade-off between size and number of reversals.

References

1. Berman, P., Lingas, A.: On complexity of regular languages in terms of finite automata. Report 304, Institute of Computer Science, Polish Academy of Sciences, Warsaw (1977)
2. Geffert, V., Mereghetti, C., Pighizzini, G.: Converting two-way nondeterministic unary automata into simpler automata. Theoretical Computer Science 295, 189–203 (2003)
3. Geffert, V., Pighizzini, G.: Two-way unary automata versus logarithmic space. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 197–208. Springer, Heidelberg (2010)
4. Hromkovič, J.: Descriptional complexity of finite automata: concepts and open problems. Journal of Automata, Languages and Combinatorics 7(4), 519–531 (2002)
5. Hromkovič, J., Schnitger, G.: Nondeterminism versus determinism for two-way finite automata: generalizations of Sipser's separation. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 439–451. Springer, Heidelberg (2003)
6. Kapoutsis, C.: Small sweeping 2NFAs are not closed under complement. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 144–156. Springer, Heidelberg (2006)
7. Kapoutsis, C.: Deterministic moles cannot solve liveness. Journal of Automata, Languages and Combinatorics 12(1-2), 215–235 (2007)
8. Kapoutsis, C.: Two-way automata versus logarithmic space. In: Proceedings of Computer Science in Russia, pp. 359–372 (2011)
9. Leung, H.: Tight lower bounds on the size of sweeping automata. Journal of Computer and System Sciences 63(3), 384–393 (2001)
10. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two-way finite automata. In: Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, pp. 275–286. ACM, San Diego (1978)
11. Seiferas, J.I.: Untitled manuscript, communicated to M. Sipser (October 1973)
12. Sipser, M.: Lower bounds on the size of sweeping automata. Journal of Computer and System Sciences 21(2), 195–202 (1980)

Isomorphism of Regular Trees and Words*

Markus Lohrey and Christian Mathissen

Institut für Informatik, Universität Leipzig, Germany
{lohrey,mathissen}@informatik.uni-leipzig.de

Abstract. The complexity of the isomorphism problem for regular trees, regular linear orders, and regular words is analyzed. A tree is regular if it is isomorphic to the prefix order on a regular language. In case regular languages are represented by NFAs (DFAs), the isomorphism problem for regular trees turns out to be EXPTIME-complete (resp. P-complete). In case the input automata are acyclic NFAs (acyclic DFAs), the corresponding trees are (succinctly represented) finite trees, and the isomorphism problem turns out to be PSPACE-complete (resp. P-complete). A linear order is regular if it is isomorphic to the lexicographic order on a regular language. A polynomial time algorithm for the isomorphism problem for regular linear orders (and even regular words, which generalize the latter) given by DFAs is presented. This solves an open problem by Ésik and Bloom. A long version of this paper can be found in [18].

1 Introduction

Isomorphism problems for infinite but finitely presented structures are an active research topic in algorithmic model theory [1]. It is a folklore result in computable model theory that the isomorphism problem for computable structures (i.e., structures, where the domain is a computable set of natural numbers and all relations are computable too) is highly undecidable — more precisely, it is Σ^1_1-complete, i.e., complete for the first existential level of the analytical hierarchy. Khoussainov et al. proved in [15] that even for automatic structures (i.e., structures, where the domain is a regular set of words and all relations can be recognized by synchronous multitape automata), the isomorphism problem is Σ^1_1-complete. In [16], this result was further improved to automatic order trees (trees viewed as partial orders) and automatic linear orders. On the decidability side, Courcelle proved that the isomorphism problem for equational graphs is decidable [7]. Recall that a graph is equational if it is the least solution of a system of equations over the HR graph operations. We remark that Courcelle's algorithm for the isomorphism problem for equational graphs has very high complexity (it is not elementary), since it uses the decidability of monadic second-order logic on equational graphs.

In this paper, we continue the investigation of isomorphism problems for infinite but finitely presented structures at the lower end of the spectra. We focus on two very simple classes of infinite structures: *regular trees* and *regular words*; both are particular automatic structures. Recall that a countable tree is regular if it has only finitely many subtrees up to isomorphism. This definition works for ordered trees (where the children

* This work is supported by the DFG research project ALKODA.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 210–221, 2011.

of a node are linearly ordered) and unordered trees. An equivalent characterization in the unordered case uses regular languages: An unordered (countable) tree T is regular if and only if there is a regular language $L \subseteq \Sigma^*$ which contains the empty word and such that T is isomorphic to the tree obtained by taking the prefix order on L (ε is the root). Hence, a regular tree can be represented by a finite deterministic or nondeterministic automaton (DFA or NFA), and the isomorphism problem for regular trees becomes the following computational problem: Given two DFAs (resp., NFAs) accepting both the empty word, are the corresponding regular trees isomorphic?

It is is not difficult to prove that this problem can be solved in polynomial time if the two input automata are assumed to be DFAs; the algorithm is very similar to the well-known partition refinement algorithm for checking bisimilarity of finite state systems [14]. Hence, the isomorphism problem for regular trees that are represented by NFAs can be solved in exponential time. Our first main result states that this problem is in fact EXPTIME-complete (Thm. 3.3). The proof of the EXPTIME lower bound uses three main ingredients: (i) EXPTIME coincides with alternating polynomial space [5], (ii) a construction from [13], which reduces the evaluation problem for Boolean expressions to the isomorphism problem for (finite) trees, and (iii) a small NFA accepting all words that do *not* represent an accepting computation of a polynomial space machine [23].

Our proof technique yields another result too: It is PSPACE-complete to check for two given *acyclic* NFAs $\mathcal{A}_1$, $\mathcal{A}_2$ (both accepting the empty word), whether the trees that result from the prefix orders on $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$, respectively, are isomorphic. Note that these two trees are clearly finite (since the automata are acyclic), but the size of $L(\mathcal{A}_i)$ can be exponential in the number of states of $\mathcal{A}_i$. In this sense, acyclic NFAs can be seen as a succinct representation of finite trees. The PSPACE-upper bound for acyclic NFAs follows easily from Lindell's result [19] that isomorphism of explicitly given trees can be checked in logarithmic space.

The second part of this paper studies the isomorphism problem for *regular words*, which were introduced in [6]. A *generalized word* over a finite alphabet Σ is a countable linear order together with a Σ-coloring of the elements. A generalized word is regular if it can be obtained as the least solution (in a certain sense made precise in [6]) of a system $X_1 = t_1, \ldots, X_n = t_n$. Here, every t_i is a finite word over the alphabet $\Sigma \cup \{X_1, \ldots, X_n\}$. For instance, the system $X = abX$ defines the regular word $(ab)^\omega$.

Courcelle [6] gave an alternative characterization of regular words: A generalized word is regular if and only if it is equal to the frontier word of a finitely-branching ordered regular tree, where the leaves are colored by symbols from Σ. Here, the frontier word is obtained by ordering the leaves in the usual left-to-right order (note that the tree is ordered). Alternatively, a regular word can be represented by a DFA $\mathcal{A}$, where the set of final states is partitioned into sets F_a ($a \in \Sigma$); we call such a DFA a *partitioned DFA*. The corresponding regular word is obtained by ordering the language of $\mathcal{A}$ lexicographically and coloring a word $w \in L(\mathcal{A})$ with a if w leads from the initial state to a state from F_a.

A third characterization of regular words was provided by Heilbrunner [12]: A generalized word is regular if it can be obtained from singleton words (i.e., symbols from Σ) using the operations of concatenation, ω-power, $\overline{\omega}$-power and dense shuffle. For

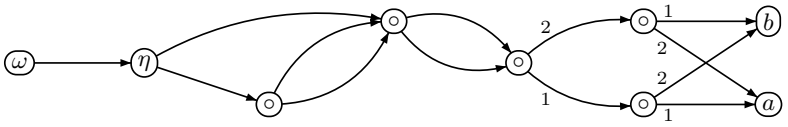


Fig. 1. A dag for the regular word $([abbaabba, abbaabbaabbaabba]^{\eta})^{\omega}$. Nodes labeled with $\circ$ compute the concatenation of their successor nodes. In case the order of the successor nodes matters, we specify it by edge labels.

a generalized word u, its ω-power (resp. $\overline{\omega}$-power) is the generalized word $uuu\cdots$ (resp. $\cdots uuu$). Moreover, the shuffle of generalized words $u_1, \ldots, u_n$ is obtained by choosing a dense coloring of the rationals with colors $\{1, \ldots, n\}$ (up to isomorphism, there is only a single such coloring [21]) and then replacing every i-colored rational by u_i. In fact, Heilbrunner presents an algorithm which computes from a given system of equations (or, alternatively, a partitioned DFA) an expression over the above set of operations (called a *regular expression* in the following) which defines the least solution of the system of equations. A simple analysis of Heilbrunner's algorithm shows that the computed regular expression in general has exponential size with respect to the input system of equations and it is easy to see that this cannot be avoided (take for instance the system $X_i = X_{i+1}X_{i+1}$ $(1 \leq i \leq n)$, $X_n = a$, which defines the finite word a^{2^n}).

The next step was taken by Thomas in [24], where he proved that the isomorphism problem for regular words is decidable. For his proof, he uses the decidability of the monadic second-order theory of linear orders; hence his proof does not yield an elementary upper bound for the isomorphism problem for regular words. Such an algorithm was later presented by Bloom and Ésik in [2], where the authors present a polynomial time algorithm for checking whether two given regular expressions define isomorphic regular words. Together with Heilbrunner's algorithm, this yields an exponential time algorithm for checking whether the least solutions of two given systems of equations (or, alternatively, the regular words defined by two partitioned DFAs) are isomorphic. It was asked in [2], whether a polynomial time algorithm for this problem exists.

Our second main result answers this question affirmatively. In fact, we prove that the problem, whether two given partitioned DFAs define isomorphic regular words, is P-complete (Cor. 4.2 and Thm. 4.4). A large part of the long version [18] of this paper deals with the polynomial time upper bound. The first step is simple. By reanalyzing Heilbrunner's algorithm, it is easily seen that from a given partitioned DFA (defining a regular word u) one can compute in *polynomial time* a *succinct representation* of a regular expression for u. This succinct representation consists of a dag (directed acyclic graph), whose unfolding is a regular expression for u. Figure 1 shows an example of such a dag. The second and main step of our proof shows that the polynomial time algorithm of Bloom and Ésik for regular expressions can be refined in such a way that it works (in polynomial time) for succinct regular expressions too. The main tool in our proof is (besides the machinery from [2]) algorithms on compressed strings (see [22] for a survey), in particular Plandowski's result that equality of strings that are represented by *straight-line programs* (i.e., context free grammars that only generate a single word) can be checked in polynomial time [20]. It is a simple observation that

an *acyclic* partitioned DFA is basically a straight-line program. Hence, we show how to extend Plandowski's polynomial time algorithm from acyclic partitioned DFAs to general partitioned DFAs.

An immediate corollary of our result is that it can be checked in polynomial time whether the lexicographic orderings on the languages defined by two given DFAs (so called regular linear orderings) are isomorphic. For the special case that the two input DFAs accept well-ordered languages, this was shown in [8]. Let us mention that it is highly undecidable (Σ^1_1-complete) to check, whether the lexicographic orderings on the languages defined by two given deterministic pushdown automata (these are the algebraic linear orderings [3]) are isomorphic [16].

2 Preliminaries

We assume standard notions from automata theory. Let us take a finite alphabet Σ. For $u, v \in \Sigma^*$, we write $u \leq_{\mathsf{pref}} v$ if there exists $w \in \Sigma^*$ with $v = uw$, i.e., u is a *prefix* of v. A language $L \subseteq \Sigma^*$ is *prefix-closed* if $u \leq_{\mathsf{pref}} v \in L$ implies $u \in L$. For a fixed linear order $\leq$ on the alphabet Σ we define the *lexicographic order* $\leq_{\mathsf{lex}}$ on Σ^* as follows: $u \leq_{\mathsf{lex}} v$ if $u \leq_{\mathsf{pref}} v$ or there exist $w, x, y \in \Sigma^*$ and $a, b \in \Sigma$ such that $a < b$, $u = wax$, and $v = wby$. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic finite automaton (NFA) where Q is the set of states, Σ is the input alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. Then, $\mathcal{A}$ is called *prefix-closed* if $Q = F$ (thus, $L(\mathcal{A})$ is prefix-closed). For a deterministic finite automaton (DFA), δ is a partial map from $Q \times \Sigma$ to Q. A *partitioned DFA* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, (F_a)_{a \in \Gamma})$, where Γ is a finite alphabet, $\mathcal{B} := (Q, \Sigma, \delta, q_0, \bigcup_{a \in \Gamma} F_a)$ is an ordinary DFA and $F_a \cap F_b = \emptyset$ for $a \neq b$. Since $\mathcal{B}$ is a DFA, it follows that the language $L(\mathcal{B})$ is partitioned by the languages $L(Q, \Sigma, \delta, q_0, F_a)$ $(a \in \Gamma)$.

We assume that the reader has some basic background in complexity theory, in particular concerning the complexity classes P, PSPACE, and EXPTIME. All completeness results in this paper refer to logspace reductions.

2.1 Trees

A *tree* is a partial order $T = (A; \leq)$, where $\leq$ has a smallest element (the root of the tree; in particular $A \neq \emptyset$) and for every $a \in A$, the set $\{b \in A \mid b \leq a\}$ is finite and linearly ordered by $\leq$. We write $a \lessdot b$ if $a < b$ and there does not exist $c \in A$ with $a < c < b$. Then $(A; \lessdot)$ is a tree in the graph theoretical sense (sometimes, it is also called a successor tree). For two trees T_1 and T_2, we write $T_1 \cong T_2$ in case T_1 and T_2 are isomorphic. A *tree over the finite alphabet* Σ is a pair $T = (L; \leq_{\mathsf{pref}})$, where $L \subseteq \Sigma^*$ is a language with $\varepsilon \in L$. Note that T is indeed a tree in the above sense (ε is the root). If L is prefix-closed, then, clearly, T is a finitely branching tree.

A countable tree T is called *regular* if T has only finitely many subtrees up to isomorphism, see e.g. [4,24]. Equivalently, a countable tree is regular if it is isomorphic to a tree of the form $(L; \leq_{\mathsf{pref}})$, where L is a regular language with $\varepsilon \in L$. If L is accepted by the DFA $\mathcal{A}$ and all final states can be reached from the initial state, then the subtrees of $(L; \leq_{\mathsf{pref}})$ correspond to the final states of $\mathcal{A}$. Note that by our definition, a regular

tree needs not be finitely branching. A finitely branching tree is regular if and only if it is the unfolding of a finite directed graph [4].

Our first definition of a regular tree (having only finitely many subtrees up to isomorphism) makes sense for other types of trees as well, e.g. for node-labeled trees or ordered trees (where the children of a node are linearly ordered). These variants of regular trees can be generated by finite automata as well. For instance, a node-labeled regular tree $(L; \leq_{\mathsf{pref}}, (L_a)_{a\in\Gamma})$, where Γ is a finite labeling alphabet and L_a is the set of a-labeled nodes can be specified by a partitioned DFA $(Q, \Sigma, \delta, q_0, (F_a)_{a\in\Gamma})$ with $L_a = L(Q, \Sigma, \delta, q_0, F_a)$ and $L = \bigcup_{a\in\Gamma} L_a$. We do not consider node labels in this paper, since it makes no difference for the isomorphism problem (node labels can be eliminated by adding additional children to nodes). Ordered regular trees are briefly discussed in the long version [18] of this paper.

2.2 Linear Orders and Generalized Words

See [21] for a thorough introduction into linear orders. Let η be the order type of the rational numbers, ω be the order type of the natural numbers, and $\overline{\omega}$ be the order type of the negative integers. With $\mathbf{n}$ we denote a finite linear order with n elements. Let $\Lambda = (L; \leq)$ be a linear order. An *interval* of Λ is a subset $I \subseteq L$ such that $x < z < y$ and $x, y \in I$ implies $z \in I$. The predecessor (resp., successor) of $x \in L$ is a largest (resp., smallest) element of $\{y \in L \mid y < x\}$ (resp., $\{y \in L \mid x < y\}$). Of course, the *predecessor* (resp., *successor*) of x need not exist, but if it exists then it is unique. The linear order Λ is *dense* if L consists of at least two elements, and for all $x < y$ there exists z with $x < z < y$. By Cantor's theorem, every countable dense linear order, which neither has a smallest nor largest element is isomorphic to η. Hence, if we take symbols 0 and 1 with $0 < 1$, then $(\{0,1\}^*1; \leq_{\mathsf{lex}}) \cong \eta$. The linear order Λ is *scattered* if there does not exist an injective order morphism $\varphi : \eta \to \Lambda$. Clearly, ω, $\overline{\omega}$, as well as every finite linear order are scattered. A linear order is *regular* if it is isomorphic to a linear order $(L; \leq_{\mathsf{lex}})$ for a regular language L. For instance, ω, $\overline{\omega}$, η, and every finite linear order are regular linear orders.

Generalized words are countable colored linear orders. Let Σ be a finite alphabet. A *generalized word* (or simply word) u over Σ is a triple $(L; \leq, \tau)$ such that L is a finite or countably infinite set, $\leq$ is a linear order on L and $\tau : L \to \Sigma$ is a coloring of L. The alphabet $\mathsf{alph}(u)$ equals the image of τ. If L is finite, we obtain a finite word in the usual sense. Moreover, $u = (L; \leq, \tau)$ is scattered if $(L; \leq)$ is scattered. We write $u \cong v$ for generalized words u and v, if u and v are isomorphic.

There is a natural operation of concatenation of two generalized words. Let $u_1 = (L_1; \leq_1, \tau_1)$ and $u_2 = (L_2; \leq_2, \tau_2)$ be generalized words with $L_1 \cap L_2 = \emptyset$. Then u_1u_2 is the generalized word $(L_1 \cup L_2; \leq, \tau_1 \cup \tau_2)$, where $x \leq y$ if and only if either $x, y \in L_1$ and $x \leq_1 y$, or $x, y \in L_2$ and $x \leq_2 y$, or $x \in L_1$ and $y \in L_2$. Similarly, we can define the ω-power (resp., $\overline{\omega}$-power) of a generalized word u as the generalized word that results from ω (resp. $\overline{\omega}$) by replacing every point by a copy of u. So, intuitively, $u^\omega = uuu\cdots$ and $u^{\overline{\omega}} = \cdots uuu$. Finally, we need the shuffle operator. Given generalized words $u_1, \ldots, u_n$, we let $[u_1, \ldots, u_n]^\eta$ be the generalized word that is obtained from η as follows: Take a coloring of η with colors $1, \ldots, n$ such that for all $x, y \in \mathbb{Q}$ with $x < y$ and all $1 \leq i \leq n$, there exists $x < z < y$ such that z has color i (it can

be shown that up to isomorphism there is a unique such dense coloring [21]). Then the *shuffle of* $u_1, \ldots, u_n$, denoted by $[u_1, \ldots, u_n]^\eta$, is obtained by replacing every i-colored rational by a copy of the generalized word u_i. Since $[u_1, \ldots, u_n]^\eta$ is invariant under permutations of the u_i, we also sometimes use the notation X^η for a finite set X of generalized words. The least set of words which contains the singleton words a for $a \in \Sigma$ and is closed under concatenation, ω-power, $\overline{\omega}$-power, and shuffle is called the set of *regular words* over Σ, denoted $\mathrm{Reg}(\Sigma)$. Note that this definition implies that every regular word is non-empty, i.e., its domain is a non-empty set. Clearly, every regular word can be described by a *regular expression* over the above operations, but this regular expression is in general not unique. Given a regular expression e, we define the corresponding regular word by $\mathsf{val}(e)$.

By a result of Heilbrunner [12], regular words can be characterized by partitioned DFAs as follows: Let $\mathcal{A} = (Q, \Gamma, \delta, q_0, (F_a)_{a\in\Sigma})$ be a partitioned DFA, and let $\mathcal{B} = (Q, \Gamma, \delta, q_0, \bigcup_{a\in\Sigma} F_a)$. Let us fix a linear order on the alphabet Γ, so that the lexicographic order $\leq_{\mathsf{lex}}$ is defined on Γ^*. Then we denote with $w(\mathcal{A})$ the generalized word $w(\mathcal{A}) = (L(\mathcal{B}); \leq_{\mathsf{lex}}, \tau)$, where $\tau(u) = a$ ($a \in \Sigma$, $u \in L(\mathcal{B})$) if and only if $u \in L(Q, \Gamma, \delta, q_0, F_a)$. It is easy to construct from a given regular expression e a partitioned DFA $\mathcal{A}$ with $\mathsf{val}(e) \cong w(\mathcal{A})$, see e.g. [24, proof of Prop. 2] for a simple construction. The other direction is more difficult. Heilbrunner has shown in [12] how to compute from a given partitioned DFA $\mathcal{A}$ (such that $w(\mathcal{A})$ is non-empty) a regular expression e with $\mathsf{val}(e) \cong w(\mathcal{A})$.[1] Unfortunately, the size of the regular expression produced by Heilbrunner's algorithm is exponential in the size of $\mathcal{A}$. On the other hand, reanalyzing Heilbrunner's algorithm shows that a succinct representation of a regular expression for $w(\mathcal{A})$ can be produced in polynomial time. This succinct representation is a dag (directed acyclic graph), where multiple occurrences of the same regular subexpression are represented only once. We denote such dags with $\mathbb{A}, \mathbb{B}$, etc. The regular word represented by the dag $\mathbb{A}$ is again denoted by $\mathsf{val}(\mathbb{A})$.

3 Isomorphism Problem for Regular Trees

In this section, we investigate the isomorphism problem for (unordered) regular trees. We consider two input representations for regular trees: DFAs and NFAs. It turns out that while the isomorphism problem for DFA-represented regular trees is P-complete, the same problem becomes EXPTIME-complete for NFA-represented regular trees. Moreover, we show that for *finite* trees that are succinctly represented by *acyclic* NFAs, isomorphism is PSPACE-complete.

Let us start with upper bounds. Our proof of the following theorem is based on an algorithm similar to the partition refinement algorithm for checking bisimilarity of finite state systems [14]. The statement for NFAs clearly follows from the statement for DFAs using the powerset construction for transforming NFAs into DFAs.

Theorem 3.1. *For two given DFAs (resp., NFAs) $\mathcal{A}_1$, $\mathcal{A}_2$ such that $\varepsilon \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ one can decide in polynomial time (resp., exponential time) whether $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$.*

[1] In fact, Heilbrunner [12] speaks about systems of equations and their least solutions instead of partitioned DFAs. These two formalisms can be efficiently transformed into each other.

For acyclic NFAs, we can improve the upper bound from Thm. 3.1 to PSPACE.

Theorem 3.2. *For two given acyclic NFAs $\mathcal{A}_1$, $\mathcal{A}_2$ such that $\varepsilon \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ one can decide in polynomial space whether $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$.*

The proof of Thm. 3.2 is based on two facts: (i) Given an acyclic NFA $\mathcal{A}$ we can compute an explicit representation of the finite tree $(L(\mathcal{A}); \leq_{\mathsf{pref}})$ (using e.g. adjacency lists) by a transducer, whose working tape is polynomially bounded, and (ii) isomorphism for explicitly given finite trees can be checked in logspace [19].

Concerning lower bounds, our main result is:

Theorem 3.3. *It is* EXPTIME*-hard (and hence* EXPTIME*-complete) to decide for two given prefix-closed NFAs $\mathcal{A}_1$, $\mathcal{A}_2$, whether $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$.*

It is straightforward to prove PSPACE-hardness of the problem in Thm. 3.3. If Σ is the underlying alphabet of a given NFA $\mathcal{A}$, then $(L(\mathcal{A}); \leq_{\mathsf{pref}})$ is a full $|\Sigma|$-ary tree if and only if $L(\mathcal{A}) = \Sigma^*$. But universality for NFAs is PSPACE-complete [23]. The proof for the EXPTIME lower bound stated in Thm. 3.3 is more involved. Here is a rough outline: EXPTIME coincides with alternating polynomial space [5]. Checking whether a given input is accepted by a polynomial space bounded alternating Turing machine M amounts to evaluate a Boolean expression whose gates correspond to configurations of M. Using a construction from [13], the evaluation problem for (finite) Boolean expressions can be reduced to the isomorphism problem for (finite) trees. In our case, the Boolean expression will be infinite. Nevertheless, the infinite Boolean expressions we have to deal with can be evaluated because on every infinite path that starts in the root (the output gate) there is either an and-gate, where one of the inputs is a false-gate, or an or-gate, where one of the inputs is a true-gate. Applying the construction from [13] to an infinite Boolean expression (that arises from our construction) yields two infinite trees, which are isomorphic if and only if our infinite Boolean expression evaluates to true. Luckily, these two trees turn out to be regular, and they can be represented by small NFAs. Using a similar construction, but starting with an alternating polynomial time machine (instead of an alternating polynomial space machine), we can prove:

Theorem 3.4. *It is* PSPACE*-hard (and hence* PSPACE*-complete) to decide for two given prefix-closed acyclic NFAs $\mathcal{A}_1$, $\mathcal{A}_2$, whether $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$.*

Finally, by a reduction from the P-complete monotone circuit value problem [11] (which uses again the reduction from the evaluation problem for Boolean expressions to the isomorphism problem for explicitly given finite trees [13]), we get the next result.

Theorem 3.5. *It is* P*-hard (and hence* P*-complete) to decide for two given prefix-closed acyclic DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$, whether $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$.*

4 Isomorphism Problem for Regular Words

In this section we study the isomorphism problem for regular words that are represented by partitioned DFAs. We prove that this problem as well as the isomorphism problem for regular linear orders that are represented by DFAs are P-complete. It follows that the isomorphism problem for regular linear orders that are represented by NFAs can be solved in exponential time. We show that this problem is also PSPACE-hard.

4.1 Upper Bounds

In Section 2.2 we mentioned that Heilbrunner's algorithm [12] transforms a given partitioned DFA $\mathcal{A}$ into a succinct regular expression $\mathbb{A}$ (in form of a dag) for the regular word $w(\mathcal{A})$. This motivates the following result:

Theorem 4.1. *For two given dags $\mathbb{A}_1$ and $\mathbb{A}_2$ one can decide in polynomial time, whether* $\mathsf{val}(\mathbb{A}_1) \cong \mathsf{val}(\mathbb{A}_2)$.

The next result is an immediate corollary of Thm. 4.1 and [12].

Corollary 4.2. *For two given partitioned DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ one can decide in polynomial time whether $w(\mathcal{A}_1) \cong w(\mathcal{A}_2)$.*

Our proof of Thm. 4.1 is quite long and technical. But essentially, we use the same strategy as in [2]. Recall that Bloom and Ésik prove in [2] that for two given (non-succinct) regular expressions e_1, e_2 it can be decided in polynomial time, whether they represent the same regular word. Let us briefly explain their strategy.

A central concept in [2] is the notion of a block of a generalized word. Blocks allow to condensate a generalized word to a coarser word (whose elements are the blocks of the original word). Let $u = (L; \leq, \tau)$ be a generalized word over the alphabet Σ. A *subword* of u is an interval I of the linear order $(L; \leq))$ together with the coloring τ restricted to I. A *uniform subword* of u is a subword that is isomorphic to a shuffle Γ^η for some $\Gamma \subseteq \Sigma$. A uniform subword is a *maximal uniform subword* if it is not properly contained in another uniform subword. Now let v be a subword such that no point of v is contained in a uniform subword of u. Then v is *successor-closed* if for each point p of v, whenever the successor and the predecessor of p exist, they are contained in v as well. A successor-closed subword is minimal if it does not strictly contain another successor-closed subword. Finally, a *block* of the generalized word u is either a maximal uniform subword of u or a minimal successor-closed subword of u. A regular word which consists of a single block is called *primitive*.[2] By [2] a generalized word u is primitive if and only if it is of one of the following forms (where $x, z \in \Sigma^+$, $y \in \Sigma^*$): A finite non-empty word, a scattered word of the form $x^{\overline{\omega}}y$, a scattered word of the form yz^ω, a scattered word of the form $x^{\overline{\omega}}yz^\omega$, or a uniform word ($\Gamma^\eta$ for some $\Gamma \subseteq \Sigma$). Let $D(\Sigma)$ be the set of all primitive words over Σ.

Let u be a regular word. Each point p of u belongs to some unique block $\mathrm{Bl}(p)$, which induces a regular (and hence primitive) word. Moreover we can order the blocks of u linearly by setting $\mathrm{Bl}(p) < \mathrm{Bl}(q)$ if and only if $p < q$. The order obtained that way is denoted $(\mathrm{Bl}(u); \leq)$. Then we extend the order $(\mathrm{Bl}(u); \leq)$ to a generalized word $\widehat{u}$ over the alphabet $D(\Sigma)$, called the *skeleton* of u, by labeling each block with the corresponding isomorphic word in $D(\Sigma)$. Implicitly, it is shown in [2] that for every regular word u there exists a *finite* subset of $D(\Sigma)$ such that every block of u is isomorphic to a generalized word from that finite subset. Moreover, $\widehat{u}$ is a regular word over that finite subset of $D(\Sigma)$. Bloom and Ésik have shown that two regular words are isomorphic if and only if their skeletons are isomorphic [2, Cor. 73].

[2] In combinatorics on words, a finite word is called primitive, if it is not a proper power of a non-empty word. Our notion of a primitive word should not be confused with this definition.

From two given regular expression e_1 and e_2, Bloom and Ésik compute in polynomial time two "simpler" expressions f_1 and f_2 (over a finite alphabet, which consists of a finite subset of $D(\Sigma)$) such that $\mathsf{val}(f_i)$ is the skeleton of $\mathsf{val}(e_i)$. Here, "simpler" means that the height of the expression f_i is strictly smaller than the height of e_i. The above mentioned Cor. 73 from [2] allows to replace e_1, e_2 by f_1, f_2. This step is iterated until one of the two expressions denotes a primitive regular word. If at that point the other expression does not denote a primitive word, then the two initial regular words are not isomorphic. On the other hand, if both regular expressions denote primitive regular words, then one faces the problem of checking whether two given primitive regular words are isomorphic. It is straightforward to do this in polynomial time.

For succinct expressions (i.e., dags), we use the same strategy. Given two dags $\mathbb{A}_1$ and $\mathbb{A}_2$, we compute in polynomial time new dags $\mathbb{B}_1$ and $\mathbb{B}_2$ such that (i) $\mathsf{val}(\mathbb{B}_i)$ is the skeleton of $\mathsf{val}(\mathbb{A}_i)$ and (ii) the height of $\mathbb{B}_i$ is strictly smaller than the height of $\mathbb{A}_i$. Note that the notion of "height" makes sense for dags as well. It is the maximal length of a path in the dag. To obtain a polynomial time algorithm at the end, several problems have to be addressed. First of all, the transformation of a dag $\mathbb{A}$ into a dag $\mathbb{B}$ such that $\mathsf{val}(\mathbb{B})$ is the skeleton of $\mathsf{val}(\mathbb{A})$ must be accomplished in polynomial time. But even if we can achieve this, an overall polynomial running time is not guaranteed, since we have to iterate this transformation. If for instance, the size of $\mathbb{B}$ (let us define the size of a dag as the number of edges) would be twice the size of $\mathbb{A}$, then this would result into an overall exponential blow-up. But fortunately, our transformation of $\mathbb{A}$ into $\mathbb{B}$ only involves an additive blow-up, which is polynomial at each iteration. Finally, at the end, we have to check isomorphism for two primitive regular words that are succinctly represented by dags. It is not obvious to do this in polynomial time. In fact, our algorithm for solving this problem makes essential use of known results for compressed words.

Let us explain this in more detail. A dag, where only the alphabet symbols and the operation of concatenation is used (no ω- and $\overline{\omega}$-powers and no shuffles) is also known as a *straight-line program (SLP)*. Alternatively, it can be seen as an acyclic context-free grammar, where each nonterminal is the left-hand side of a unique production. Such a context-free grammar generates a single finite word. Moreover, the length of this word can be exponential in the size of the SLP. Hence the SLP can be seen as a compressed representation of the finite word. In recent years, a lot of effort was spent on the development of efficient algorithms for SLP-represented finite words: Our polynomial time algorithm for primitive regular words that are given by dags uses a seminal result from this area: It can be checked in polynomial time, whether the word represented by a first SLP is a factor of the word represented by a second SLP (compressed pattern matching). The best algorithm for compressed pattern matching has a cubic running time [17]. Note that as a corollary, it can be checked in polynomial time, whether two given SLPs represent the same finite word. This result was first shown by Plandowski [20].

4.2 Lower Bounds for Regular Linear Orders

Let us now turn to lower bounds for the isomorphism problem for regular words. In fact, all these lower bounds already hold for a unary alphabet, i.e., they hold for regular linear orders. The results in this section nicely contrast the results from Section 3, where we studied the isomorphism problem for the prefix order trees on regular languages. In this section, we replace the prefix order by the lexicographical order.

Theorem 4.3. *For every finite alphabet Σ, it is* P*-hard (and hence* P*-complete) to decide for two given dags $\mathbb{A}_1$ and $\mathbb{A}_2$ over the alphabet Σ, whether* $\mathsf{val}(\mathbb{A}_1) \cong \mathsf{val}(\mathbb{A}_2)$.

As for the proof of Thm. 3.5, the proof of Thm. 4.3 is based on a reduction from the monotone circuit value problem. We do not know, whether the lower bound from Thm. 4.3 holds for ordinary expressions (instead of dags) too.

Theorem 4.4. *It is* P*-hard (and hence* P*-complete) to decide for two given DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$, whether* $(L(\mathcal{A}_1); \leq_{\mathsf{lex}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{lex}})$.

Proof. Note that by Cor. 4.2 the problem belongs to P. For P-hardness, it suffices by Thm. 4.3 to construct in logspace from a given dag $\mathbb{A}$ (over a unary terminal alphabet) a DFA $\mathcal{A}$ such that the linear order $\mathsf{val}(\mathbb{A})$ is isomorphic to $(L(\mathcal{A}); \leq_{\mathsf{lex}})$. But this is accomplished by the construction in the proof of [24, Prop. 2]. □

Cor. 4.2 implies that it can be checked in EXPTIME whether the lexicographical orderings on two regular languages, given by NFAs, are isomorphic. We do not know whether this upper bound is sharp. Currently, we can only prove a lower bound of PSPACE:

Theorem 4.5. *It is* PSPACE*-hard to decide for two given NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$, whether* $(L(\mathcal{A}_1); \leq_{\mathsf{lex}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{lex}})$.

Proof. We prove PSPACE-hardness by a reduction from the PSPACE-complete problem whether a given NFA $\mathcal{A}$ over the terminal alphabet $\{a, b\}$ accepts $\{a, b\}^*$ [23]. So let $\mathcal{A}$ be an NFA over the terminal alphabet $\{a, b\}$ and let $K = L(\mathcal{A})$. Let $\Sigma = \{0, 1, a, b, \$_1, \$_2\}$ and fix the following order on Σ: $\$_1 < 0 < 1 < \$_2 < a < b$. Under this order, $(\{0, 1\}^*1; \leq_{\mathsf{lex}}) \cong (\{a, b\}^*b; \leq_{\mathsf{lex}}) \cong \eta$.

It is straightforward to construct from $\mathcal{A}$ in logspace an NFA for the language

$$L \;=\; (\{a, b\}^*b\,\$_1) \;\cup\; (K\,b\,\{0, 1\}^*1) \;\cup\; (\{a, b\}^*b\,\$_2). \tag{1}$$

It follows that

$$(L; \leq_{\mathsf{lex}}) \cong \sum_{w \in \{a,b\}^*b} \mathcal{L}(w) \quad \text{with} \quad \mathcal{L}(w) \cong \begin{cases} \mathbf{1} + \eta + \mathbf{1} & \text{if } w \in K \\ \mathbf{2} & \text{else.} \end{cases}$$

(the sum is taken over all words from $\{a, b\}^*b$ in lexicographic order). If $K \neq \{a, b\}^*$, then $(L; \leq_{\mathsf{lex}})$ contains an interval isomorphic to $\mathbf{2}$. Hence $(L; \leq_{\mathsf{lex}}) \ncong \eta$. On the other hand, if $K = \{a, b\}^*$, then $(L; \leq_{\mathsf{lex}}) \cong (\mathbf{1}+\eta+\mathbf{1})\cdot\eta \cong \eta$. This proves the theorem. □

The proof of Thm. 4.5 shows that it is PSPACE-hard to check for a given NFA $\mathcal{A}$, whether $(L(\mathcal{A}); \leq_{\mathsf{lex}}) \cong \eta$. In fact, this problem turns out to be PSPACE-complete, see the long version [18] for details.

In [9] it is shown that the problem, whether $(L; \leq_{\mathsf{lex}}) \cong \eta$ for a given context-free language, is undecidable. This result is shown by a reduction from Post's correspondence problem. Note that this result can be also easily deduced using the technique from the above proof: If we start with a pushdown automaton for $\mathcal{A}$ instead of an NFA, then the language L from (1) is context-free. Hence, $(L; \leq_{\mathsf{lex}}) \cong \eta$ if and only if $L(\mathcal{A}) = \{a, b\}^*$. The latter property is a well-known undecidable problem.

Table 1. Main results for the isomorphism problem for regular trees

	DFA	NFA
acyclic	P-complete	PSPACE-complete
arbitrary		EXPTIME-complete

Table 2. Main results for the isomorphism problem for regular linear orders

	DFA	NFA
acyclic	$\mathsf{C}_{=}\mathsf{L}$-complete	$\mathsf{C}_{=}\mathsf{P}$-complete
arbitrary	P-complete	PSPACE-hard, in EXPTIME

In Section 3 we also studied the isomorphism problem for finite trees that are succinctly given by the prefix order on the finite language accepted by an acyclic DFA (resp., NFA). To complete the picture, we should also consider the isomorphism problem for linear orders that consist of a lexicographically ordered finite language, where the latter is represented by an acyclic DFA (resp., NFA). Of course, this problem is somehow trivial, since two finite linear orders are isomorphic if and only if they have the same cardinality. Hence, we have to consider the problem whether two given acyclic DFAs (resp. NFAs) accept languages of the same cardinality. The complexity of these problems is analyzed in the long version [18]. Straightforward arguments show that checking whether two acyclic DFAs (resp. NFAs) accept languages of the same cardinality is complete for the counting class $\mathsf{C}_{=}\mathsf{L}$ (resp., $\mathsf{C}_{=}\mathsf{P}$), see [18] for definitions.

5 Conclusion and Open Problems

Table 1 (Table 2) summarizes our complexity results for the isomorphism problem for regular trees (regular linear orders). Let us conclude with some open problems. As can be seen from Table 2, there is a complexity gap for the isomorphism problem for regular linear orders that are represented by NFAs. This problem belongs to EXPTIME and is PSPACE-hard. Another interesting problem concerns the equivalence problem for straight-line programs (i.e., dags that generate finite words, or equivalently, acyclic partitioned DFAs, or equivalently, context-free grammars that generate a single word). Plandowski has shown that this problem can be solved in polynomial time. Recall that this result is fundamental for our polynomial time algorithm for dags (Thm. 4.1). In [10], it was conjectured that the equivalence problem for straight-line programs is P-complete, but this is still open.

References

1. Bárány, V., Grädel, E., Rubin, S.: Automata-based presentations of infinite structures. In: Finite and Algorithmic Model Theory. London Mathematical Society Lecture Notes Series, vol. 379. Cambridge University Press, Cambridge (to appear, 2011)

2. Bloom, S.L., Ésik, Z.: The equational theory of regular words. Inf. Comput. 197(1-2), 55–89 (2005)
3. Bloom, S.L., Ésik, Z.: Algebraic linear orderings. Technical report, arXiv.org (2010), http://arxiv.org/abs/1002.1624
4. Caucal, D.: On infinite terms having a decidable monadic theory. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 165–176. Springer, Heidelberg (2002)
5. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM 28(1), 114–133 (1981)
6. Courcelle, B.: Frontiers of infinite trees. ITA 12(4) (1978)
7. Courcelle, B.: The definability of equational graphs in monadic second-order logic. In: Ronchi Della Rocca, S., Ausiello, G., Dezani-Ciancaglini, M. (eds.) ICALP 1989. LNCS, vol. 372, pp. 207–221. Springer, Heidelberg (1989)
8. Ésik, Z.: Representing small ordinals by finite automata. In: Proc. DCFS 2010. EPTCS, vol. 31, pp. 78–87 (2010)
9. Ésik, Z.: An undecidable property of context-free linear orders. Inf. Process. Lett. 111(3), 107–109 (2011)
10. Gasieniec, L., Gibbons, A., Rytter, W.: Efficiency of fast parallel pattern searching in highly compressed texts. In: Kutyłowski, M., Wierzbicki, T., Pacholski, L. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 48–58. Springer, Heidelberg (1999)
11. Goldschlager, L.M.: The monotone and planar circuit value problems are log space complete for P. SIGACT News 9(2), 25–99 (1977)
12. Heilbrunner, S.: An algorithm for the solution of fixed-point equations for infinite words. ITA 14(2), 131–141 (1980)
13. Jenner, B., Köbler, J., McKenzie, P., Torán, J.: Completeness results for graph isomorphism. J. Comput. Syst. Sci. 66(3), 549–566 (2003)
14. Kanellakis, P.C., Smolka, S.A.: CCS expressions, finite state processes, and three problems of equivalence. Inf. Comput. 86(1) (1990)
15. Khoussainov, B., Nies, A., Rubin, S., Stephan, F.: Automatic structures: richness and limitations. Logical Methods in Computer Science 3(2), 18(electronic) (2007)
16. Kuske, D., Liu, J., Lohrey, M.: The isomorphism problem on classes of automatic structures with transitive relations. submitted for publication, extended version of a paper presented at LICS 2010 (2011)
17. Lifshits, Y.: Processing compressed texts: A tractability border. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 228–240. Springer, Heidelberg (2007)
18. Lohrey, M., Mathissen, C.: Isomorphism of regular trees and words. Technical report, arXiv.org (2011), http://arxiv.org/abs/1102.2782
19. Lindell, S.: A logspace algorithm for tree canonization (extended abstract). In: Proc. STOC 1992, pp. 400–404. ACM Press, New York (1992)
20. Plandowski, W.: Testing equivalence of morphisms on context-free languages. In: van Leeuwen, J. (ed.) ESA 1994. LNCS, vol. 855, pp. 460–470. Springer, Heidelberg (1994)
21. Rosenstein, J.: Linear Ordering. Academic Press, London (1982)
22. Rytter, W.: Grammar compression, LZ-encodings, and string algorithms with implicit input. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 15–27. Springer, Heidelberg (2004)
23. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time (preliminary report). In: Proc. STOCS 1973, pp. 1–9. ACM Press, New York (1973)
24. Thomas, W.: On frontiers of regular trees. ITA 20(4), 371–381 (1986)

On the Capabilities of Grammars, Automata, and Transducers Controlled by Monoids

Georg Zetzsche

Fachbereich Informatik, Technische Universität Kaiserslautern,
Postfach 3049, 67653 Kaiserslautern, Germany
zetzsche@cs.uni-kl.de

Abstract. During recent decades, classical models in language theory have been extended by control mechanisms defined by monoids. We study which monoids cause the extensions of context-free grammars, finite automata, or finite state transducers to exceed the capacity of the original model. Furthermore, we investigate when, in the extended automata model, the nondeterministic variant differs from the deterministic one in capacity. We show that all these conditions are in fact equivalent and present an algebraic characterization. In particular, the open question of whether every language generated by a valence grammar over a finite monoid is context-free is provided with a positive answer.

1 Introduction

The idea of equipping classical models of theoretical computer science with a monoid (or a group) as a control mechanism has been pursued in recent decades by several authors [4,7,8,9,11,13,14]. This interest is justified by the fact that these extensions allow for a uniform treatment of a wide range of automata and grammar models: Suppose a storage mechanism can be regarded as a set of states on which a set of partial transformations operates and a computation is considered valid if the composition of the executed transformations is the identity. Then, this storage constitutes a certain monoid control.

For example, in a pushdown storage, the operations *push* and *pop* (for each participating stack symbol) and compositions thereof are partial transformations on the set of words over some alphabet. In this case, a computation is considered valid if, in the end, the stack is brought back to the initial state, i.e., the identity transformation has been applied. As further examples, blind and partially blind multicounter automata (see [5]) can be regarded as finite automata controlled by a power of the integers and of the bicyclic monoid (see [13]), respectively.

Another reason for studying monoid controlled automata, especially in the case of groups, is that the word problems of a group G are contained in a full trio (such as the context-free or the indexed languages) if and only if the languages accepted by valence automata over G are contained in this full trio (see, for example, [8, Proposition 2]). Thus, valence automata offer an automata theoretic interpretation of word problems for groups.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 222–233, 2011.

A similar situation holds for context-free grammars where each production is assigned a monoid element such that a derivation is valid as soon as the product of the monoid elements (in the order of the application of the rules) is the identity. Here, the integers, the multiplicative group of $\mathbb{Q}$, and powers of the bicyclic monoid lead to additive and multiplicative valence grammars and Petri net controlled grammars, respectively. The latter are in turn equivalent to matrix grammars (with erasing and without appearance checking, see [2] for details). Therefore, the investigation of monoid control mechanisms promises very general insights into a variety of models.

One of the most basic problems regarding these models is the characterization of those monoids whose use as control mechanism actually increases the power of the respective model. For monoid controlled automata, such a characterization has been achieved by Mitrana and Stiebe [9] for the case of groups. The author of this work was informed by an anonymous referee that a characterization for arbitrary monoids had been found by Render [12]. For valence grammars, that is, context-free grammars with monoid control, very little was known in this respect up to date. It was an open problem whether valence grammars over finite monoids are capable of generating languages that are not context-free[1] (see [4, p. 387]).

Another important question considers for which monoids the extended automata can be determinized, that is, for which monoids the deterministic variant is as powerful as the nondeterministic one. Mitrana and Stiebe [9] have shown that automata controlled by a group cannot be determinized if the group contains at least one element of infinite order. However, the exact class of monoids for which automata can be determinized was not known to date.

The contribution of this work is twofold. On the one hand, the open question of whether all languages generated by valence grammars over finite monoids are context-free is settled affirmatively. On the other hand, we use an algebraic dichotomy of monoids to provide a characterization for all the conditions above. Specifically, we show that the following assertions are equivalent:

- Valence grammars over M generate only context-free languages.
- Valence automata over M accept only regular languages.
- Valence automata over M can be determinized.
- Valence transducers over M perform only rational transductions.
- In each finitely generated submonoid of M, only finitely many elements possess a right inverse.

Note that the equivalence of the second and the last assertion has been established independently by Render [12].

2 Basic Notions

A *monoid* is a set M together with an associative operation and a neutral element. Unless defined otherwise, we will denote the neutral element of a monoid

[1] Note, however, that *valence grammars with target sets* over finite monoids are known to generate all matrix languages[3].

by 1 and its operation by juxtaposition. That is, for a monoid M and $a, b \in M$, $ab \in M$ is their product. The *opposite monoid* M^{op} of M has the same set of elements as M, but has the operation $\circ$ with $a \circ b := ba$, $a, b \in M$. For $a, b \in M$, we write $a \sqsubseteq b$ iff there are $c, d \in M$ such that $b = ac = da$. Let $a \in M$. An element $b \in M$ with $ab = 1$ is called a *right inverse* of a. If $b \in M$ obeys $ba = 1$, it is a *left inverse* of a. An element that is both a left and a right inverse is said to be a *two-sided inverse*. By $\mathbf{1}$, we denote the trivial monoid that consists of just one element. M is said to be *left-cancellative* if $ab = ac$ implies $b = c$ for $a, b, c \in M$. Whenever M^{op} is left-cancellative, we say that M is *right-cancellative*.

A subset $N \subseteq M$ is said to be a *submonoid of* M iff $1 \in N$ and $a, b \in N$ implies $ab \in N$. For a subset $N \subseteq M$, let $\langle N \rangle$ be the intersection of all submonoids N' of M that contain N. That is, $\langle N \rangle$ is the smallest submonoid of M that contains N. $\langle N \rangle$ is also called the *submonoid generated by* N. We call a monoid *finitely generated* if it is generated by a finite subset. In each monoid M, we have the following submonoids:

$$\mathfrak{R}(M) := \{a \in M \mid \exists b \in M : ab = 1\},$$
$$\mathfrak{L}(M) := \{a \in M \mid \exists b \in M : ba = 1\}.$$

The elements of $\mathfrak{R}(M)$ and $\mathfrak{L}(M)$ are called *right invertible* and *left invertible*, respectively. In addition, for every element $a \in M$, we define the sets

$$\overrightarrow{\mathfrak{I}}(a) := \{b \in M \mid ab = 1\},$$
$$\overleftarrow{\mathfrak{I}}(a) := \{b \in M \mid ba = 1\}.$$

When using a monoid M as part of a control mechanism, the subset

$$\mathfrak{E}(M) := \{a \in M \mid \exists b, c \in M : bac = 1\}$$

will play an important role. If in M every element has a two-sided inverse, we call M a *group*.

Let Σ be a fixed countable set of abstract symbols, the finite subsets of which are called *alphabets*. For an alphabet X, we will write X^* for the set of words over X. The empty word is denoted by $\lambda \in X^*$. In particular, $\emptyset^* = \{\lambda\}$. Together with the concatenation as its operation, X^* is a monoid. We will regard every $x \in X$ as an element of X^*, namely the word consisting of only one occurence of x. For a symbol $x \in X$ and a word $w \in X^*$, let $|w|_x$ be the number of occurrences of x in w. For a subset $Y \subseteq X$, let $|w|_Y := \sum_{x \in Y} |w|_x$. By $|w|$, we will refer to the length of w. By $X^{\leq n} \subseteq X^*$, for $n \in \mathbb{N}$, we denote the set of all words over X of length $\leq n$. Given alphabets X, Y, subsets of X^* and $X^* \times Y^*$ are called *languages* and *transductions*, respectively. We define the *shuffle* $L_1 \sqcup\!\sqcup L_2$ of two languages $L_1, L_2 \subseteq X^*$ to be the set of all words $w \in X^*$ such that $w = u_1 v_1 \cdots u_n v_n$ for some $u_i, v_i \in X^*$, $1 \leq i \leq n$, with $u_1 \cdots u_n \in L_1$, $v_1 \cdots v_n \in L_2$. When $\{w\}$ is used as an operand for $\sqcup\!\sqcup$, we also just write w instead of $\{w\}$. For $x_1, \ldots, x_n \in X$, let $(x_1 \cdots x_n)^{\mathrm{rev}} := x_n \cdots x_1$.

Let M be a monoid. An *automaton over* M is a tuple $A = (M, Q, E, q_0, F)$, in which Q is a finite set of *states*, E is a finite subset of $Q \times M \times Q$ called the

set of *edges*, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of *final states*. The *step relation* $\Rightarrow_A$ of A is a binary relation on $Q \times M$, for which $(p, a) \Rightarrow_A (q, b)$ iff there is an edge (p, c, q) such that $b = ac$. The set generated by A is then

$$S(A) := \{a \in M \mid \exists q \in F : (q_0, 1) \Rightarrow_A^* (q, a)\}.$$

A *valence automaton over* M is an automaton A over $X^* \times M$, where X is an alphabet. A is said to be *deterministic* if all its edges are in $Q \times (X \times M) \times Q$ and, for each pair $(q, x) \in Q \times X$, there is at most one edge $(q, (x, m), p)$ for $m \in M, p \in Q$. The *language accepted by* A is defined as

$$L(A) := \{w \in X^* \mid (w, 1) \in S(A)\}.$$

A *finite automaton* is a valence automaton over $\mathbf{1}$. For a finite automaton $A = (X^* \times \mathbf{1}, Q, E, q_0, F)$, we also write $A = (X, Q, E, q_0, F)$. Languages accepted by finite automata are called *regular languages*. A *valence transducer over* M is an automaton A over $X^* \times Y^* \times M$, where X and Y are alphabets. The *transduction performed by* A is

$$T(A) := \{(x, y) \in X^* \times Y^* \mid (x, y, 1) \in S(A)\}.$$

A *finite state transducer* is a valence transducer over $\mathbf{1}$. For a finite state transducer $A = (X^* \times Y^* \times \mathbf{1}, Q, E, q_0, F)$, we also write $A = (X, Y, Q, E, q_0, F)$. Transductions performed by finite state transducers are called *rational transductions*.

A *valence grammar over* M is a tuple $G = (N, T, M, P, S)$, where N, T are disjoint alphabets called the *nonterminal* and *terminal alphabet*, respectively, $P \subseteq N \times (N \cup T)^* \times M$ is a finite set of *productions*, and $S \in N$ is the *start symbol*. For a production $(A, w, m) \in P$, we also write $(A \to w; m)$. The *derivation relation* $\Rightarrow_G$ of G is a binary relation on $(N \cup T)^* \times M$, for which $(u, a) \Rightarrow_G (v, b)$ iff there is a $(A \to w; c) \in P$ and words $r, s \in (N \cup T)^*$ such that $u = rAs$, $v = rws$, and $b = ac$. The *language generated by* G is defined as

$$L(G) := \{w \in T^* \mid (S, 1) \Rightarrow_G^* (w, 1)\}.$$

Valence grammars were introduced by Păun in [11]. A thorough treatment, including normal form results and a classification of the resulting language classes for commutative monoids, has been carried out by Fernau and Stiebe [4]. Valence grammars over $\mathbf{1}$ are called *context-free grammars*. For a context-free grammar $G = (N, T, \mathbf{1}, P, S)$, we also write $G = (N, T, P, S)$. Furthermore, a production $(A \to w; 1) \in P$ in a context-free grammar is also written $A \to w$. Languages generated by context-free grammars are called *context-free*.

3 A Dichotomy of Monoids

The results in this section have been obtained by the author, but he was made aware by an anonymous referee that they are well-known to semigroup theorists.

See, for example, [12, Corollary 4.1.8] or [1, Chapter 2]. The proofs by the author are included, since they are elementary and therefore accessible to a broader audience.

An *infinite ascending chain* in M is an infinite sequence $x_1, x_2, \ldots$ of pairwise distinct elements of M such that $x_i \sqsubseteq x_{i+1}$ for all $i \in \mathbb{N}$.

Lemma 1. *Let M be left- or right-cancellative. Then, exactly one of the following holds:*

1. *M is a finite group.*
2. *M contains an infinite ascending chain.*

Proof. Suppose M does not contain an infinite ascending chain.

First, we prove that M is a group. Let $r \in M$ and consider the elements $r^i \in M$, $i \in \mathbb{N}$. Since $r^i \sqsubseteq r^j$ for $i \leq j$, our assumption implies that there are $i, j \in \mathbb{N}$, $i < j$, with $r^i = r^j$. Since M is left- or right-cancellative, this implies $r^{j-i} = 1$, meaning that r has in r^{j-i-1} a two-sided inverse. Thus, M is a group.

This implies that $r \sqsubseteq s$ for any $r, s \in M$. Therefore, if M were infinite, it would contain an infinite ascending chain. □

Lemma 2. *Let $s, t \in M$, $s \neq t$, and $s \sqsubseteq t$. Then, $\overrightarrow{\mathfrak{I}}(s) \cap \overrightarrow{\mathfrak{I}}(t) = \emptyset$ and $\overleftarrow{\mathfrak{I}}(s) \cap \overleftarrow{\mathfrak{I}}(t) = \emptyset$.*

Proof. We only show $\overrightarrow{\mathfrak{I}}(s) \cap \overrightarrow{\mathfrak{I}}(t) = \emptyset$ since then $\overleftarrow{\mathfrak{I}}(s) \cap \overleftarrow{\mathfrak{I}}(t) = \emptyset$ follows by applying the former to the opposite monoid. Write $t = us$, $u \in M$, and suppose there were a $z \in \overrightarrow{\mathfrak{I}}(s) \cap \overrightarrow{\mathfrak{I}}(t)$. Then, $1 = tz = usz = u$ and thus $t = s$. □

Theorem 1. *For every monoid M, exactly one of the following holds:*

1. *The subsets $\mathfrak{R}(M)$, $\mathfrak{L}(M)$, and $\mathfrak{E}(M)$ coincide and constitute a finite group.*
2. *$\mathfrak{R}(M)$ and $\mathfrak{L}(M)$ each contain an infinite ascending chain. In particular, there exist infinite sets $S \subseteq \mathfrak{R}(M)$ and $S' \subseteq \mathfrak{L}(M)$ such that $\overrightarrow{\mathfrak{I}}(s) \cap \overrightarrow{\mathfrak{I}}(t) = \emptyset$ for $s, t \in S$, $s \neq t$, and $\overleftarrow{\mathfrak{I}}(s') \cap \overleftarrow{\mathfrak{I}}(t') = \emptyset$ for $s', t' \in S'$, $s' \neq t'$.*

Proof. First, we claim that $\mathfrak{R}(M)$ is infinite if and only if $\mathfrak{L}(M)$ is infinite. Here, it suffices that $\mathfrak{R}(M)$ being infinite implies the infinity of $\mathfrak{L}(M)$, since the other direction follows by considering the opposite monoid. If $\mathfrak{R}(M)$ is infinite, it contains an infinite ascending chain according to Lemma 1. By Lemma 2, the elements of the chain have pairwise disjoint sets of right inverses that are non-empty. Since right inverses are left invertible, $\mathfrak{L}(M)$ is infinite.

Suppose $\mathfrak{R}(M)$ and $\mathfrak{L}(M)$ are both finite. Since $\mathfrak{R}(M)$ is right-cancellative, it is a group by Lemma 1. Thus, we have $\mathfrak{R}(M) \subseteq \mathfrak{L}(M)$ and analogously $\mathfrak{L}(M) \subseteq \mathfrak{R}(M)$. In order to prove $\mathfrak{E}(M) = \mathfrak{R}(M)$, we observe that $\mathfrak{R}(M) \subseteq \mathfrak{E}(M)$ by definition. Now, suppose $a \in \mathfrak{E}(M)$ to be witnessed by $bac = 1$, $b, c \in M$. By this equation, we have $b \in \mathfrak{R}(M)$ and can multiply b^{-1} on the left and then b on the right. We obtain $acb = 1$ and thus $a \in \mathfrak{R}(M)$. This proves that $\mathfrak{R}(M) = \mathfrak{L}(M) = \mathfrak{E}(M)$ and that this is a finite group.

In case $\mathfrak{R}(M)$ and $\mathfrak{L}(M)$ are both infinite, the infinite ascending chains are provided by Lemma 1. By Lemma 2, their elements form sets $S \subseteq \mathfrak{R}(M)$ and $S' \subseteq \mathfrak{L}(M)$ with the desired properties. □

4 Capabilities of Valence Automata and Transducers

In this section, we show that the following conditions are equivalent:

- Valence automata over M accept only regular languages.
- Valence automata over M can be determinized.
- Valence transducers over M perform only rational transductions.
- $\mathfrak{R}(N)$ is finite for every finitely generated submonoid N of M.

The equivalence of the first and the last condition has been obtained independently by Render [12].

Using Theorem 1, it is not difficult to prove the following lemma. Therefore, we omit the proof and refer the reader to [12, Theorem 4.1.10] and [4, Theorem 4.4, Lemma 4.7].

Lemma 3. *Let $\mathfrak{R}(N)$ be finite for every finitely generated submonoid N of M. Then, valence automata over M accept only regular languages and valence transducers over M perform only rational transductions. In particular, valence automata over M can be determinized.*

In [9], Mitrana and Stiebe proved that valence automata over groups with at least one element of infinite order cannot be determinized. We can now use a similar idea and the dichotomy of monoids to provide a characterization of those monoids over which valence automata can be determinized.

Lemma 4. *Let M be a finitely generated monoid such that $\mathfrak{R}(M)$ is infinite. Then, there is a valence automaton over M whose accepted language cannot be accepted by a deterministic valence automaton over M. In particular, valence automata over M can accept non-regular languages and valence transducers over M can perform non-rational transductions.*

Proof. Let M be generated by the finite set $\{a_1, \ldots, a_n\}$ and let $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$ be disjoint alphabets. Let $\varphi : (X \cup Y)^* \to M$ be the epimorphism defined by $\varphi(x_i) := \varphi(y_i) := a_i$ and $K := X^* \cup \{w \in X^*Y^* \mid \varphi(w) = 1\}$. Then, K is clearly accepted by a (nondeterministic) valence automaton over M. Suppose K were accepted by a deterministic valence automaton A over M. Let $S \subseteq \mathfrak{R}(M)$ be the infinite set provided by Theorem 1. The infinity of S implies that we can find an infinite set $S' \subseteq X^*$ such that $\varphi(S') = S$ and $\varphi(u) \neq \varphi(v)$ for $u, v \in S'$, $u \neq v$. Since A is deterministic and $S' \subseteq L(A)$, each word $w \in S'$ causes A to enter a configuration $(q(w), 1)$ where $q(w)$ is a final state. Choose $u, v \in S'$ such that $u \neq v$ and $q(u) = q(v)$. Let $u' \in Y^*$ be a word such that $\varphi(u)\varphi(u') = 1$. This is possible since $\varphi(u) \in \mathfrak{R}(M)$ and φ is surjective. The word u' causes A to go from $(q(u), 1) = (q(v), 1)$ to $(q, 1)$ for some final state q, since $uu' \in K$. Thus, vu' is also contained in K and hence $\varphi(v)\varphi(u') = 1$, but $\overrightarrow{\mathfrak{I}}(\varphi(u)) \cap \overrightarrow{\mathfrak{I}}(\varphi(v)) = \emptyset$, a contradiction.

Hence, K is not accepted by a deterministic valence automaton over M. In particular, K is not regular. Furthermore, from the valence automaton accepting K, a valence transducer can be constructed that maps $\{\lambda\}$ to K. Since K is not regular, the transduction performed by the transducer is not rational. □

5 Capabilities of Valence Grammars

In this section, it is shown that the following conditions are equivalent:

- Valence grammars over M generate only context-free languages.
- $\mathfrak{R}(N)$ is finite for every finitely generated submonoid N of M.

In one of the directions, we have to construct a context-free grammar for valence grammars over monoids that fulfill the second condition. Because of the limited means available in the context-free case, the constructed grammar can simulate only a certain fragment of the derivations in the valence grammar. Thus, we will have to make sure that every word generated by the valence grammar has a derivation in the aforementioned fragment. These derivations are obtained by considering the derivation tree of a given derivation and then choosing a suitable linear extension of the tree order. The construction of these linear extensions can already be described for a simpler kind of partial order, *valence trees*.

Let X be an alphabet and $U \subseteq X$ a subset. Then, each word $w \in X^*$ has a unique decomposition $w = y_0 x_1 y_1 \cdots x_n y_n$ such that $y_0, y_n \in (X \setminus U)^*$, $y_i \in (X \setminus U)^+$ for $1 \leq i \leq n-1$, and $x_i \in U^+$ for $1 \leq i \leq n$. This decomposition is called the *U-decomposition* of w and we define $\rho(w, U) := n$.

A *tree* is a finite partially ordered set $(\mathcal{T}, \leq)$ that has a least element and where, for each $t \in \mathcal{T}$, the set $\{t' \in \mathcal{T} \mid t' \leq t\}$ is totally ordered by $\leq$. The least element is also called the *root* and the maximal elements are called *leaves*. A *valence tree* $\mathcal{T}$ over M is a tuple $(\mathcal{T}, \leq, \varphi)$, where $(\mathcal{T}, \leq)$ is a tree and $\varphi : \mathcal{T}^* \to M$ is a homomorphism[2] assigning a *valence* to each node. An *evaluation* defines an order in which the nodes in a valence tree can be traversed that is compatible with the tree order. Thus, an *evaluation* of $\mathcal{T}$ is a linear extension $\preceq$ of $(\mathcal{T}, \leq)$. Let $w \in \mathcal{T}^*$ correspond to $\preceq$, i.e., let $\mathcal{T} = \{t_1, \ldots, t_n\}$ such that $t_1 \preceq \cdots \preceq t_n$ and $w = t_1 \cdots t_n$. Then, the *value* of $\preceq$ is defined to be $\varphi(w)$. An element $v \in M$ is called a *value of* $\mathcal{T}$ if there exists an evaluation of $(\mathcal{T}, \leq)$ with value v. Given a node $t \in \mathcal{T}$, let $U_t := \{t' \in \mathcal{T} \mid t \leq t'\}$. If $w = y_0 x_1 y_1 \cdots x_n y_n$ is the U_t-decomposition of w, then $\varphi(x_1), \ldots, \varphi(x_n)$ is called the *valence sequence* of t in w and n its *length*. By the *excursiveness* of an evaluation, we refer to the maximal length of a valence sequence. Hence, the excursiveness of an evaluation is the maximal number of times one has to enter any given subtree when traversing the nodes in the order given by the evaluation. We are interested in finding evaluations of valence trees with small excursiveness. Of course, for every valence tree, there are evaluations with excursiveness one (take, for example, the order induced by a preorder traversal), but these might not be able to cover all possible values. However, we will see in Lemma 7 that, in the case of a finite group, there exists a bound m such that every value can be attained by an evaluation of excursiveness of at most m.

Lemma 5. *For each finite group G, there is a constant $m \in \mathbb{N}$ with the following property: For elements $g_i, h_i \in G$, $i = 1, \ldots, n$, $n \geq m$, there are indices $k, \ell \in \mathbb{N}$, $1 \leq k < \ell \leq n$, such that $g_k h_k \cdots g_\ell h_\ell = g_k \cdots g_\ell h_k \cdots h_\ell$.*

[2] We will often assume, without loss of generality, that $\mathcal{T}$ is an alphabet.

Proof. Let $m = 2(|G|^3 + 1)$ and $D \subseteq \{1, \ldots, n\}$ be the set of odd indices. Define the map $\alpha : D \to G^3$ by $\alpha(i) := (g_1 \cdots g_i,\ h_1 \cdots h_i,\ g_1 h_1 \cdots g_i h_i)$ for $i \in D$. Since $|D| \geq |G|^3 + 1$, there are indices $i, j \in D$, $i < j$, such that $\alpha(i) = \alpha(j)$. This means that $g_{i+1} \cdots g_j = 1$, $h_{i+1} \cdots h_j = 1$, and $g_{i+1} h_{i+1} \cdots g_j h_j = 1$. Since i, j are both odd, letting $k = i + 1$ and $\ell = j$ implies $k < \ell$ and yields the desired equality. □

The proof of the following technical lemma can be done using a case analysis and is not included due to space restrictions.

Lemma 6. *Let X be an alphabet and $U, V \subseteq X$ subsets such that either $U \subseteq V$, $V \subseteq U$, or $U \cap V = \emptyset$. Furthermore, let $r \in X^* U$, $x \in U^+$, $y \in (X \setminus U)^+$, and $s \in X^* \setminus U X^*$. Then, we have $\rho(rxys, V) \leq \rho(ryxs, V)$.*

Lemma 7. *For each finite group G, there is a constant m such that each value of a valence tree over G has an evaluation of excursiveness of at most m.*

Proof. For an alphabet X, we denote the set of *multisets* over X, i.e., maps $X \to \mathbb{N}$, by $X^{\oplus}$. $X^{\oplus}$ carries a (commutative) monoid structure by way of $(\alpha + \beta)(x) := \alpha(x) + \beta(x)$ for $x \in X$. To every evaluation w of $(\mathcal{T}, \leq)$, we assign the multiset $\mu_w \in \mathcal{T}^{\oplus}$ that is defined by $\mu_w(t) := \rho(w, U_t)$ for every $t \in \mathcal{T}$. That is, $\mu_w(t)$ is the length of the valence sequence of t in w.

Let m be the constant provided by Lemma 5 and let $w \in \mathcal{T}^*$ be an evaluation of $(\mathcal{T}, \leq)$ such that μ_w is minimal with respect to $\sqsubseteq$ among all evaluations with the value v. If we can prove that $\mu_w(t) \leq m$ for all $t \in \mathcal{T}$, the lemma follows. Therefore, suppose that there is a $t \in \mathcal{T}$ with $n := \mu_w(t) > m$. Specifically, let $w = y_0 x_1 y_1 \cdots x_n y_n$ be the U_t-decomposition of w. Use Lemma 5 to find indices $1 \leq k < \ell \leq n$ with

$$\varphi(x_k)\varphi(y_k) \cdots \varphi(x_\ell)\varphi(y_\ell) = \varphi(x_k) \cdots \varphi(x_\ell)\varphi(y_k) \cdots \varphi(y_\ell). \tag{1}$$

Furthermore, let

$$w' := (y_0 x_1 y_1 \cdots x_{k-1} y_{k-1})(x_k \cdots x_\ell y_k \cdots y_\ell)(x_{\ell+1} y_{\ell+1} \cdots x_n y_n). \tag{2}$$

That is, we obtain w' from w by replacing $x_k y_k \cdots x_\ell y_\ell$ with $x_k \cdots x_\ell y_k \ldots y_\ell$. Then, (1) means that $\varphi(w') = \varphi(w)$. We shall prove that w' is an evaluation of $(\mathcal{T}, \leq)$ and obeys $\mu_{w'} \sqsubset \mu_w$, which contradicts the choice of w.

First, we prove that w' is an evaluation. Let $u_1, u_2 \in \mathcal{T}$ be nodes with $u_1 \leq u_2$. If $u_1 < t$, then u_1 appears in y_0, and thus u_2 is on the right side of u_1 in w'. If $u_1 \geq t$, then each of the nodes u_1, u_2 appears in some x_i and therefore do not change their relative positions. If u_1 and t are incomparable, then u_2 and t are also incomparable and each of u_1, u_2 appears in some y_i. Again, u_1 and u_2 do not change their relative positions. Thus, w' corresponds to a linear extension of $\leq$.

We want to show that $\mu_{w'} \sqsubseteq \mu_w$. To this end, we consider the words

$$w_i := (y_0 x_1 y_1 \cdots x_{k-1} y_{k-1})(x_k \cdots x_{k+i} y_k \cdots y_{k+i})(x_{k+i+1} y_{k+i+1} \cdots x_n y_n)$$

for $0 \leq i \leq \ell - k$. With these, we have $w = w_0$ and $w' = w_{\ell-k}$. Since $(\mathcal{T}, \leq)$ is a tree, we have $U_u \subseteq U_t$, $U_t \subseteq U_u$, or $U_u \cap U_t = \emptyset$ for every $u \in \mathcal{T}$. Therefore, we can apply Lemma 6 to $U := U_t$, $V := U_u$, and

$$r := (y_0 x_1 y_1 \cdots x_{k-1} y_{k-1})(x_k \cdots x_{k+i}), \quad x := x_{k+i+1},$$
$$y := y_k \cdots y_{k+i}, \qquad\qquad s := y_{k+i+1}(x_{k+i+2} y_{k+i+2} \cdots x_n y_n),$$

which yields $\rho(w_{i+1}, U_u) \leq \rho(w_i, U_u)$ for $0 \leq i < \ell - k$. This implies $\mu_{w'}(u) \leq \mu_w(u)$ and therefore $\mu_{w'} \sqsubseteq \mu_w$.

It remains to be shown that $\mu_{w'}$ is strictly smaller than μ_w. In w', the node t has the valence sequence

$$\varphi(x_1), \ldots, \varphi(x_{k-1}), \varphi(x_k \cdots x_\ell), \varphi(x_{\ell+1}), \cdots \varphi(x_n),$$

which has length $\mu_{w'}(t) = n - (\ell - k) < n = \mu_w(t)$. □

We define a *derivation tree* for a valence grammar $G = (N, T, M, P, S)$ to be a tuple $(\mathcal{T}, \leq, \varphi, (\leq_t)_{t \in \mathcal{T}}, \Lambda)$, where

- $(\mathcal{T}, \leq, \varphi)$ is a valence tree,
- for each $t \in \mathcal{T}$, $\leq_t$ is a total order on the set of successors of t,
- $\Lambda : \mathcal{T} \to N \cup T \cup \{\lambda\}$ defines a *label* for each node,
- if $t \in \mathcal{T}$ is a node with the successors $s_1, \ldots, s_n$ such that $s_1 \leq_t \ldots \leq_t s_n$, then we either have $\Lambda(t) \in T \cup \{\lambda\}$, $n = 0$, and $\varphi(t) = 1$ or we have $\Lambda(t) \in N$ and there is a production $(\Lambda(t) \to \Lambda(s_1) \cdots \Lambda(s_n); \varphi(t))$ in P.

The total orders $\leq_t$, $t \in \mathcal{T}$, induce a total order on the set of leaves (see [6, Section 4.3] for details), which in turn defines a word $w \in T^*$. This word is called the *yield* of the derivation tree.

Each derivation tree can be regarded as a valence tree. An evaluation then defines a derivation $(A, 1) \Rightarrow_G^* (w, v)$, where $A \in N$ is the label of the root, w is the yield, and $v \in M$ is the value of the evaluation. Conversely, every derivation induces a derivation tree and an evaluation. Thus, a word $w \in T^*$ is in $L(G)$ iff there exists a derivation tree for G with yield w, a root labeled S, and an evaluation with value 1. See [4, Section 4.2] for details.

Lemma 8. *Let $\mathfrak{R}(N)$ be finite for every finitely generated submonoid N of M. Furthermore, let $G = (N, T, M, P, S)$ be a valence grammar over M. Then, $L(G)$ is context-free.*

Proof. We can assume that M is finitely generated and thus has a finite $\mathfrak{R}(M)$. Since productions $(A \to w; m)$ with $m \notin \mathfrak{E}(M)$ cannot be part of a successful derivation, their removal does not change the generated language. Furthermore, by Theorem 1, $\mathfrak{E}(M)$ is a finite group. Thus, we can assume that $G = (N, T, H, P, S)$, where $H = \mathfrak{E}(M)$ is a finite group. By a simple construction, we can further assume that in G, every production is of the form $(A \to w; h)$ with $w \in N^*$ or $(A \to w; 1)$ with $w \in T \cup \{\lambda\}$.

We shall construct a context-free grammar $G' = (N', T, P', S')$ for $L(G)$. The basic idea is that G' will simulate derivations of bounded excursiveness.

This is done by letting the nonterminals in G' consist of a nonterminal $A \in N$ and a finite sequence σ of elements from H. G' then simulates the generation of a nonterminal A by generating a pair (A, σ) and thereby guesses that the corresponding node in the derivation tree of G will have σ as its valence sequence. Lemma 7 will then guarantee that this allows G' to derive all words in $L(G)$ when the sequences σ are of bounded length.

Formally, we will regard H as an alphabet and a sequence will be a word over H. In order to be able to distinguish between the concatenation of words in H^* and the group operation in H, we will denote the concatenation in H^* by $\square$. Let $N' = N \times H^{\leq m}$, in which $m \in \mathbb{N}$ is the constant provided by Lemma 7 for the group H. The set of sequences that can be obtained from another sequence σ by "joining" subsequences is denoted by $J(\sigma)$:

$$J(h_1 \square h_2 \square \sigma) := J((h_1 h_2) \square \sigma) \cup \{h_1 \square \sigma' \mid \sigma' \in J(h_2 \square \sigma)\}$$

for $h_1, h_2 \in H$ and $\sigma \in H^*$ and $J(\sigma) := \{\sigma\}$ if $|\sigma| \leq 1$. J is defined for subsets $S \subseteq H^*$ by $J(S) := \bigcup_{\sigma \in S} J(\sigma)$.

For each production $(A \to w; h) \in P$, $w = B_1 \cdots B_n$, $B_i \in N$ for $1 \leq i \leq n$, we include the production

$$(A, \sigma) \to (B_1, \sigma_1) \cdots (B_n, \sigma_n),$$

for each $\sigma \in H^{\leq m} \setminus \{\lambda\}$ and $\sigma_1, \ldots, \sigma_n \in H^{\leq m}$ such that for $\sigma = h_1 \square \sigma'$, $h_1 \in H$, $\sigma' \in H^{\leq m-1}$, one of the following holds:

- $(h^{-1} h_1) \square \sigma' \in J(\sigma_1 \sqcup \cdots \sqcup \sigma_n)$.
- $h_1 = h$ and $\sigma' \in J(\sigma_1 \sqcup \cdots \sqcup \sigma_n)$.

Furthermore, for every production $(A \to w, 1)$, $w \in T \cup \{\lambda\}$, we include $(A, \lambda) \to w$. Finally, the start symbol of G' is $(S, 1)$.

It remains to be shown that $L(G') = L(G)$. In order to prove $L(G') \subseteq L(G)$, one can show by induction on n that for $w \in T^*$, $(A, \sigma) \Rightarrow_{G'}^n w$ implies that there is a derivation $(A, 1) \Rightarrow_G^* (w, h)$ for some $h \in H$ using productions $(A_1 \to w_1; h_1), \ldots, (A_k \to w_k; h_k)$ such that $\sigma \in J(h_1 \square \cdots \square h_k)$. This implies that for $(S, 1) \Rightarrow_{G'}^* w$, $w \in T^*$, we have $w \in L(G)$. Thus, $L(G') \subseteq L(G)$.

Let $w \in L(G)$ with derivation tree $(\mathcal{T}, \leq, \varphi, (\leq_t)_{t \in \mathcal{T}}, \Lambda)$. By Lemma 7, there is an evaluation $\preceq$ of the tree of excursiveness $\leq m$. From the tree and the evaluation, we construct a derivation tree $(\mathcal{T}, \leq, \varphi', (\leq_t)_{t \in \mathcal{T}}, \Lambda')$ for w in G' as follows. The components $\mathcal{T}$, $\leq$, and $\leq_t$, $t \in \mathcal{T}$, remain unaltered, but φ' will assign 1 to each node and Λ' is defined by $\Lambda'(t) := \Lambda(t)$ if $\Lambda(t) \in T \cup \{\lambda\}$ and $\Lambda'(t) := (\Lambda(t), h_1 \square \cdots \square h_k)$ if $\Lambda(t) \in N$, where $h_1, \ldots, h_k$ is the valence sequence of t in $\preceq$. Now, one can see that the new tree is a derivation tree for G' that generates w with any evaluation. Hence, $L(G) \subseteq L(G')$. □

In order to prove the main result of this section, we need to exhibit a valence grammar over M that generates a non-context-free language when given a finitely generated monoid M with infinite $\mathfrak{R}(M)$. In the proof that the generated language is not context-free, we will use the following well-known Iteration Lemma by Ogden [10].

Lemma 9 (Ogden). *For each context-free language L, there is an integer m such that for any word $z \in L$ and any choice of at least m distinct marked positions in z, there is a decomposition $z = uvwxy$ such that:*

1. *w contains at least one marked position.*
2. *Either u and v both contain marked positions, or x and y both contain marked positions.*
3. *vwx contains at most m marked positions.*
4. *$uv^iwx^iy \in L$ for every $i \geq 0$.*

Lemma 10. *Let $\mathfrak{R}(M)$ be infinite for some finitely generated monoid M. Then, there is a valence grammar over M that generates a language that is not context-free.*

Proof. Let M be generated by $a_1, \ldots, a_n$ and let $X = \{x_1, \ldots, x_n\}$ be an alphabet. Furthermore, let $\varphi : X^* \to M$ be the surjective homomorphism defined by $\varphi(x_i) = a_i$. The valence grammar $G = (N, T, M, P, S_0)$ is defined as follows. Let $N = \{S_0, S_1\}$, $T = X \cup \{c\}$, and let P consist of the productions

$$(S_0 \to x_i S_0 x_i, a_i), \quad (S_0 \to cS_1c, 1), \quad (S_1 \to x_i S_1, a_i), \quad (S_1 \to \lambda, 1)$$

for $1 \leq i \leq n$. Then, clearly $L(G) = K := \{rcscr^{\mathrm{rev}} \mid r, s \in X^*,\ \varphi(rs) = 1\}$. It remains to be shown that K is not context-free. Suppose K is context-free and let m be the constant provided by Lemma 9. By Theorem 1, we can find an infinite subset $S \subseteq \mathfrak{L}(M)$ such that $\overleftarrow{\mathfrak{I}}(a) \cap \overleftarrow{\mathfrak{I}}(b) = \emptyset$ for $a, b \in S$, $a \neq b$. Since φ is surjective, we can define $\ell(a)$ for every $a \in S$ to be the minimal length of a word $w \in X^*$ such that $\varphi(w)a = 1$. If $\ell(a) < m$ for all $a \in S$, the finite set $\{\varphi(w) \mid w \in X^*, |w| < m\}$ contains a left inverse for every $a \in S$. This, however, contradicts the fact that the infinitely many elements of S have disjoint sets of left inverses. Thus, there exists an $a \in S$ with $\ell(a) \geq m$. We choose words $r, s \in X^*$ such that $\varphi(s) = a$ and r is of minimal length among those words satisfying $\varphi(rs) = 1$. Then, by the choice of a, we have $|r| \geq m$.

We apply the Iteration Lemma to the word $z = rcscr^{\mathrm{rev}} \in K$, where we choose the first $|r|$ symbols to be marked. Let $z = uvwxy$ be the decomposition from the lemma. Condition 1 implies $|uv| < |r|$. Because of 4, x cannot contain a c. Furthermore, x cannot be a subword of r, since then pumping would lead to words with mismatching first and third segments. In particular, from condition 2, the first part holds and v is not empty. Thus, if x were a subword of s, pumping would again lead to a mismatching first and third segment. Hence, x is a subword of r^{rev}. If we now pump with $i = 0$, we obtain a word $r'cscr'' \in K$, where $|r'| < |r|$. In particular, we have $\varphi(r's) = 1$, in contradiction to the choice of r. □

Theorem 2. *Let M be a monoid. The following conditions are equivalent:*

1. *Valence grammars over M generate only context-free languages.*
2. *Valence automata over M accept only regular languages.*
3. *Valence automata over M can be determinized.*

4. *Valence transducers over M perform only rational transductions.*
5. *$\mathfrak{R}(N)$ is finite for every finitely generated submonoid N of M.*

Proof. 1 is equivalent to 5 by Lemmas 10 and 8. Lemmas 4 and 3 prove that 2, 3, and 4 are each equivalent to 5. □

Acknowledgements. I would like to thank Reiner Hüchting and Klaus Madlener for discussions and helpful comments which have improved the presentation of the paper. Furthermore, I am grateful to the anonymous referee who made me aware that the equivalence of the second and last condition of Theorem 2 had been obtained independently by Elaine Render and that the dichotomy of monoids is well-known.

References

1. Clifford, A.H., Preston, G.B.: The Algebraic Theory of Semigroups, vol. 1. American Mathematical Society, Providence (1961)
2. Dassow, J., Turaev, S.: Petri net controlled grammars: the power of labeling and final markings. Romanian Journal of Information Science and Technology 12(2), 191–207 (2009)
3. Fernau, H., Stiebe, R.: Valence grammars with target sets. In: Words, Semigroups, and Transductions. World Scientific, Singapore (2001)
4. Fernau, H., Stiebe, R.: Sequential grammars and automata with valences. Theoretical Computer Science 276, 377–405 (2002)
5. Greibach, S.A.: Remarks on blind and partially blind one-way multicounter machines. Theoretical Computer Science 7(3), 311–324 (1978)
6. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (1979)
7. Ito, M., Martín-Vide, C., Mitrana, V.: Group weighted finite transducers. Acta Informatica 38, 117–129 (2001)
8. Kambites, M.: Formal languages and groups as memory. Communications in Algebra 37, 193–208 (2009)
9. Mitrana, V., Stiebe, R.: Extended finite automata over groups. Discrete Applied Mathematics 108(3), 287–300 (2001)
10. Ogden, W.: A helpful result for proving inherent ambiguity. Mathematical Systems Theory 2(3), 191–194 (1968)
11. Păun, G.: A new generative device: Valence grammars. Revue Roumaine de Mathématiques Pures et Appliquées 25, 911–924 (1980)
12. Render, E.: Rational Monoid and Semigroup Automata. PhD thesis, University of Manchester (2010)
13. Render, E., Kambites, M.: Rational subsets of polycyclic monoids and valence automata. Information and Computation 207(11), 1329–1339 (2009)
14. Render, E., Kambites, M.: Semigroup automata with rational initial and terminal sets. Theoretical Computer Science 411(7-9), 1004–1012 (2010)

The Cost of Traveling between Languages

Michael Benedikt, Gabriele Puppis, and Cristian Riveros

Department of Computer Science, Oxford University
Parks Road, Oxford OX13QD UK

Abstract. We show how to calculate the maximum number of edits per character needed to convert any string in one regular language to a string in another language. Our algorithm makes use of a local determinization procedure applicable to a subclass of distance automata. We then show how to calculate the same property when the editing needs to be done in streaming fashion, by a finite state transducer, using a reduction to mean-payoff games. We show that the optimal streaming editor can be produced in PTIME.

1 Introduction

Edit distance is a well-studied metric between strings, measuring how many operations are needed to get from one string to another. In this paper we look for natural (asymmetric) analogs for regular languages: how many edits does it require to get from a word in regular language R to a word in regular language T, in the worst case? Our notation is motivated by considering R to be a *restriction* – a constraint that the input is guaranteed to satisfy – and T to be a *target* – a constraint that we want to enforce.

In a prior work [1], we considered the basic question of whether one can get from a word in R to a word in T with a finite (uniformly bounded) number of edits. One of the main results of [1] was a characterization of the pairs (R, T) for which such a uniform bound exists.

Example 1. Consider the languages $R = a^*b^*$ and $T = a^*cb^*$. Clearly, any string in R can be converted to a string in T with at most 1 edit.

Such a bound, when it exists, shows that the language R is "quite close to being a subset of T" – the gap between strings in R and strings in T is small. However, having a uniform bound on the number of edits is a strong requirement. In this paper we look not at the absolute number of edits required to get from R to T, but rather at the *percentage* of letters that need to be edited.

Example 2. Consider the languages $R = (a + b)^*$ and $T = (ab + b)^*$. Roughly, for any pair of consecutive occurrences of the letter a in the input, we will have to perform one edit in order to ensure alternation in the output. In particular, the number of edits required to get from a string in R to a string in T is unbounded. On the other hand, it is clear that we need to edit approximately half of the letters in the worst case (i.e. a^{2n}) in order to produce a string in T.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 234–245, 2011.

We measure the gap from R to T via the worst case, over all strings $w \in R$, of the number of edits needed to bring w into T divided by the length of w. Since we want the definition to be robust to a finite number of outliers, we take the limit of this quantity as the strings are of larger and larger length – this is the *asymptotic (normalized) cost* in getting from R to T. This gives us a measure of the distortion needed to get from R to T, lying always between 0 and 1.

Our first main result is that we can determine the asymptotic cost effectively. The algorithm relies on ideas from *distance automata* [9], and in particular on an application of determinization of distance automata, closely related to Mohri's determinization procedure [7].

We then turn to the setting where editing is required to be done in streaming fashion, producing the edits immediately on seeing the input letter. We measure a streaming edit processor by the number of edits per character it requires to get from any string in R to a string in T, again looking at the limit as the string length gets large. We define the *streaming asymptotic cost* to be the optimal cost of a streaming processor. We show that this quantity can also be calculated effectively, using techniques from mean-payoff games.

Example 3. Consider $R = (a + b)\,c^*(a^+ + b^+)$ and $T = a\,c^*a^+ + b\,c^*b^+$. One can get from R to T by only editing the initial letter: so the asymptotic cost is 0. However, a streaming strategy must commit to changing the initial letter or leaving it be: if it makes the "incorrect" choice, it will have to edit an unbounded final segment; thus the streaming asymptotic cost is 1.

The above two results give us the ability to compare the cost one should pay in editing strings in R to strings in T with an arbitrary processor with the cost when we are restricted to use a streaming processor. If these are the same, it shows that streaming processors that edit strings in R to T can approximate arbitrary processors in worst-case behavior.

In summary our contributions are:

- We present an algorithm for calculating the asymptotic cost of transforming strings in regular language R to strings in regular language T, based on locally determinizing a subclass of distance automata.
- We give an algorithm for calculating the optimal asymptotic cost achieved using a streaming editing algorithm.

Related Work. The problem of finding the minimal distance of a string to a regular language was first considered by Wagner in [10], who showed that the problem could be solved by adapting the dynamic programming approach to edit distance, giving a polynomial time algorithm. Several authors have extended the definition to deal with distances between languages. Mohri [8] defines a distance function between two sets of strings, and more generally between string distributions: in the case of languages, this is the minimum distance between two strings in the two respective languages, which is appropriate for many applications. Konstantinidis [4] focuses on the minimum distance between distinct strings within the same language, giving tractable algorithms for computing it. Our notion of "cost" is quite distinct from this, since it is asymmetric in the two

languages, focusing on the maximum of the distance of a string in one language to the other language. In our prior work [1] we have given an algorithm for determining when this distance is finite; again the paper deals with the streaming and the non-streaming setting, but the techniques used for the finiteness problem, particularly in the non-streaming case, are radically different from those used for asymptotic cost analysis. Further related work in the database area is overviewed in [1].

Organization. Section 2 defines the basic problems. Section 3 studies the non-streaming case, while Section 4 deals with the streaming case. Section 5 gives conclusions. Proofs are relegated to the full paper.

2 Problem Setting

Given two words $w \in \Sigma^*$ and $u \in \Delta^*$, we denote by $\text{edit-dist}(w, u)$ the *Levenshtein distance* (henceforth, edit distance) between w and u, which is defined as the length of a shortest sequence s of edit operations (e.g., deleting a single character, modifying a single character, and inserting a single character) that transforms w into u [11]. Following Wagner [10], we lift this to define the distance of a word to a regular language T.

$$\text{edit-dist}(w, T) \ =^{\text{def}} \ \min\big\{\, \text{edit-dist}(w, u) \ : \ u \in T \big\}.$$

We are interested in quantifying how difficult it is to edit a word in one language to obtain a word in another. That is, we have finite alphabets Σ and Δ and regular languages $R \subseteq \Sigma^*$ and $T \subseteq \Delta^*$, called the *restriction* and *target* languages, respectively. We would like to edit a string that is known to belong to the restriction language into a string in the target language.

How do we measure the cost of edits needed to get from R to T? One method is to look at the largest number of edit operations needed to get into T from strings in R: that is, the supremum over $w \in R$ of $\text{edit-dist}(w, T)$. In an earlier work [1] we have studied for which pairs of languages (R, T) this cost is finite. However, many language pairs have infinite cost: the existence of a uniform bound to the number of edits is quite a strong property (see Example 2 in the introduction).

In this work we define an alternative notion of cost that looks at the *percentage* of symbols in a word that need to be edited. We define the *normalized cost* for editing a word w to a word in T as the fraction $\frac{\text{edit-dist}(w,T)}{|w|}$, that is, the ratio between the cost of editing w and its length. In order to measure the asymptotic behavior of the normalized cost, we define the *asymptotic cost* as the limit superior of the normalized cost when the length of words in the restriction tends to infinity. Formally, the asymptotic cost for two regular languages R and T is defined as

$$\mathbf{A}(R, T) \ =^{\text{def}} \ \lim_{n \to \infty} \sup \left\{ \frac{\text{edit-dist}(w,T)}{|w|} \ : \ w \in R, \ |w| \geq n \right\}$$

For the above definition to make sense, we always assume that R is infinite. It is easy to see that the asymptotic cost ranges over the interval $[0, 1]$ of the real

numbers. Indeed, for large words, one can modify and delete the letters to create shorter words in the target language and thus the resulting cost is always less than the length of the input word. We are ideally interested in computing the value $\mathbf{A}(R,T)$, provided that this number is rational.

Streaming vs non-streaming. The notion of "how much does it cost to edit a word in R to a word in T" assumes that an editing process could be any mapping from R to T (in principle, such a mapping could even fail to be computable). However, we know from [10] that there is a dynamic programming algorithm that, given a word w and a target language T represented by a deterministic finite state automaton (*DFA*) $\mathcal{T}$, computes in time $\mathcal{O}(|w|\cdot|\mathcal{T}|)$ an optimal edit sequence s such that $s(w) \in T$. In particular, this shows that optimal algorithms for editing a word can be described by functions of fairly low complexity. Sometimes it is desirable to have editing algorithms that are in even more limited classes. Perhaps the ideal case is when we can edit with a one-pass algorithm, that is, using a sequential transducer (note that we allow the transducer to have infinitely many states). Recall that a sequential transducer defines a word-to-word function; if this function happens to produce a word in T for every input $w \in R$, then we say that it is a *streaming edit strategy* for R and T. Similarly, we can consider k-lookahead transducers, with $k \in \mathbb{N}$: this type of transducer outputs words on the basis of its current state and an input $(k+1)$-character window that represents a substring of w of the form $w[i]\dots w[i+k]$, where $w[i]$ is either the i-th symbol of w, if $i \leq |w|$, or a dummy symbol $\perp$, if $i > |w|$. Accordingly, we talk about a k-lookahead streaming edit strategy.

Given a streaming edit strategy $\mathcal{S}$ for R and T and a word w, we define the cost of $\mathcal{S}$ on w to be the number of edits produced by $\mathcal{S}$ on w. Formally, letting $q_0 \xrightarrow{a_1/u_1} q_1 \xrightarrow{a_2/u_2} \dots \xrightarrow{a_n/u_n} q_n \xrightarrow{\varepsilon/u_{n+1}}$ be the run of the transducer $\mathcal{S}$ on a word $w = a_1 \dots a_n$, the *cost of* $\mathcal{S}$ *on* w, denoted $\mathrm{cost}(w,\mathcal{S})$, is the length of the final output u_{n+1} plus the sum of edit-dist(a_i,u_i) over all indices $1 \leq i \leq n$. Notice that the transducer $\mathcal{S}$ might output an additional string u_{n+1} at the end of its run in order to produce a word in the target language T. We can then define the asymptotic cost of a streaming (k-lookahead) edit strategy $\mathcal{S}$:

$$\mathbf{A}(R,\mathcal{S},T) \;=^{\mathrm{def}}\; \lim_{n\to\infty} \sup \left\{ \frac{\mathrm{cost}(w,\mathcal{S})}{|w|} \;:\; w \in R,\ |w| \geq n \right\}.$$

Finally, the *streaming* (k-lookahead) *asymptotic cost* for two languages R and T, denoted $\mathbf{SA}(R,T)$, is the *infimum* of $\mathbf{A}(R,\mathcal{S},T)$ taken over all streaming (k-lookahead) edit strategies $\mathcal{S}$ for R and T. We remark that, a priori, the infimum in the previous definition cannot be replaced by a minimum: it is conceivable that the asymptotic costs of the streaming edit strategies for R and T are arbitrary close to $\mathbf{SA}(R,T)$, but never achieve this value. In fact, in Section 4 we will show that this is not the case, as we can enforce, without loss of generality, a uniform bound to the memory of streaming edit strategies.

To stress the difference between the streaming and the non-streaming settings, we explicitly refer to the original problem as the asymptotic cost problem in the *non-streaming case*.

3 Asymptotic Cost in the Non-streaming Case

In this section, we study the problem of computing the asymptotic cost in the non-streaming setting. We begin with some background on distance automata, which will play a key role in the main characterization result.

Distance automata computing the edit cost. Intuitively, a *distance automaton* [9] is a transducer $\mathcal{D}$ that receives as input a finite word w and outputs a corresponding cost $\mathcal{D}(w)$ in $\mathbb{N} \cup \{\infty\}$. Formally, it is a tuple $\mathcal{D} = (\Sigma, Q, E, I, F)$, where Q is a finite set of states, $E \subseteq Q \times \Sigma \times \mathbb{N} \times Q$ is a finite transition relation, I and F are some initial and final conditions described by partial functions from Q to $\mathbb{N}$ and representing the costs of beginning and ending a run with certain states. A *run* of $\mathcal{D}$ on w is a sequence $\gamma = (q_0, a_1, c_1, q_1)\ (q_1, a_2, c_2, q_2)\ \dots\ (q_{n-1}, a_n, c_n, q_n)$ of pairwise adjacent transitions in E that spell the input word $w = a_1 a_2 \dots a_n$. The cost of the run γ is naturally defined by

$$\text{cost}(\gamma) \ =^{\text{def}} \ \sum_{1 \le i \le n} c_i.$$

We denote by $\mathcal{D}(w)$ the *minimum* value $I(q_0) + \text{cost}(\gamma) + F(q_n)$ among all states q_0 in the domain $\mathcal{D}om(I)$ of I, all states q_n in the domain $\mathcal{D}om(F)$ of F, and all runs γ of $\mathcal{D}$ on w that start in q_0 and end in q_n. We let $\mathcal{D}(w) = \infty$ if there are no such states q_0 and q_n, or if there is no run from q_0 to q_n.

When considering the edit distance of a word $w \in \Sigma^*$ to a regular language $T \subseteq \Delta^*$, it is fairly natural to express this value in terms of the cost computed by a distance automaton. By default, we assume that the target language T is recognized by a DFA $\mathcal{T} = (\Delta, Q, \delta, q_0, F)$ where Q is a finite set of states, $\delta \subseteq Q \times \Delta \times Q$ is a finite transition relation, q_0 and F are the initial and final set of states. Given two states p, q of $\mathcal{T}$, we let $\mathcal{T}_{p,q}$ be the DFA obtained from $\mathcal{T}$ by letting p be the new initial state and q the new unique final state. The distance automaton that computes the edit distance of a word over Σ to the target language $\mathscr{L}(\mathcal{T})$ is defined as $\mathcal{D}^{\text{edit}}_{\mathcal{T}} = (\Sigma, Q, E^{\text{edit}}, I^{\text{edit}}, F^{\text{edit}})$, where

- E^{edit} is the set of all transitions of the form (p, a, c, q), with $p, q \in Q$, $a \in \Sigma$, q reachable from p, and $c = \min\{\text{edit-dist}(a, v) \ : \ v \in \mathscr{L}(\mathcal{T}_{p,q})\}$,
- I^{edit} is the partial function that maps a state $q \in Q$ to the minimum among the values $\text{edit-dist}(\varepsilon, v)$, with $v \in \mathscr{L}(\mathcal{T}_{q_0,q})$ (if q is not reachable from the initial state q_0, then $I^{\text{edit}}(q)$ is undefined),
- F^{edit} is the partial function that maps a state $p \in Q$ to the minimum among the values $\text{edit-dist}(\varepsilon, v)$, with $v \in \bigcup_{q \in F} \mathscr{L}(\mathcal{T}_{p,q})$ (if p cannot reach a state in F, then $F^{\text{edit}}(p)$ is undefined).

One can easily show that $\mathcal{D}^{\text{edit}}_{\mathcal{T}}$ computes exactly the edit distance between a word $w \in \Sigma^*$ and $\mathscr{L}(\mathcal{T})$.

Proposition 1. *For every word $w \in \Sigma^*$, we have $\mathcal{D}^{\text{edit}}_{\mathcal{T}}(w) = \text{edit-dist}(w, \mathscr{L}(\mathcal{T}))$.*

Shortcut property and determinizable components. Distance automata of the form $\mathcal{D}^{\text{edit}}_{\mathcal{T}}$ are a proper sub-class of all distance automata. In particular, they satisfy the *shortcut property*, formalized just below. Given a symbol $a \in \Sigma$

and two states p, q of a distance automaton $\mathcal{D}$, we write $p \xrightarrow{a} q$ to denote the existence in $\mathcal{D}$ of a transition (p, a, c, q) with some cost $c \in \mathbb{N}$.

Definition 1. *A distance automaton $\mathcal{D}$ satisfies the* shortcut property *if for all symbols a, b and all states p, q, r, $p \xrightarrow{a} q \xrightarrow{b} r$ implies $p \xrightarrow{a} r$ and $p \xrightarrow{b} r$.*

The following lemma shows that, in particular, $\mathcal{D}_{\mathcal{T}}^{\text{edit}}$ satisfies the shortcut property.

Lemma 1. *For every DFA $\mathcal{T}$, $\mathcal{D}_{\mathcal{T}}^{\text{edit}}$ satisfies the shortcut property.*

We call a *strongly connected component* (SCC) of a distance automaton $\mathcal{D}$ any maximal set of mutually reachable states. Given a SCC C of $\mathcal{D}$, we denote by $\mathcal{D}|C$ the sub-automaton obtained from $\mathcal{D}$ by restricting the set of states and transitions to C and by letting the initial and final conditions map any state of C to 0. Note that the transition graph of $\mathcal{D}|C$ is a clique when $\mathcal{D}$ satisfies the shortcut property.

A crucial property entailed by the shortcut property is the following one. Consider two runs ρ and ρ' of $\mathcal{D}|C$ that spell the same word w, but end in different states q and q'. If ρ and ρ' have optimal cost among all runs on $\mathcal{D}|C$ on w that end in q and q' respectively, then one can show that the difference in cost between ρ and ρ' is uniformly bounded by a constant. This implies that we can determinize $\mathcal{D}|C$ by using a subset construction, maintaining the difference between the optimal cost of reaching each state q and the overall optimal cost; this is exactly Mohri's determinization procedure [7]. Since this difference is always uniformly bounded by a constant, we get a finite-state distance automaton:

Proposition 2. *For every distance automaton $\mathcal{D}$ that satisfies the shortcut property and every SCC C of $\mathcal{D}$, the sub-automaton $\mathcal{D}|C$ can be determinized.*

The above result allows us to denote by $\det(\mathcal{D}|C)$ some deterministic distance automaton equivalent to $\mathcal{D}|C$, namely, such that $\det(\mathcal{D}|C)(w) = \mathcal{D}|C(w)$ for all $w \in \Sigma^*$. The automaton $\det(\mathcal{D}|C)$ can be computed from $\mathcal{D}|C$ by a direct exponential-time algorithm [7].

Example 4. Consider the distance automaton $\mathcal{D}$ of Figure 1, which computes the edit distance of any word to the target language $T = (ab + b)^* a^*$. As $\mathcal{D}$ satisfies the shortcut property and consists of two SCCs C_1 and C_2, the two sub-automata $\mathcal{D}|C_1$ and $\mathcal{D}|C_2$ can be turned into equivalent deterministic distance automata $\det(\mathcal{D}|C_1)$ and $\det(\mathcal{D}|C_2)$, depicted to the right of Figure 1.

We remark that the above result does not imply that the entire distance automaton $\mathcal{D}$ is determinizable. Consider, for instance, a distance automaton that computes the edit distance of a word w to the target language $\mathscr{L}(\mathcal{T}) = a^* + b^*$. This distance is given by the symmetric difference between the number of occurrences of a and the number of occurrences of b and hence any deterministic device that computes edit-dist$(w, \mathscr{L}(\mathcal{T}))$ must use unbounded memory.

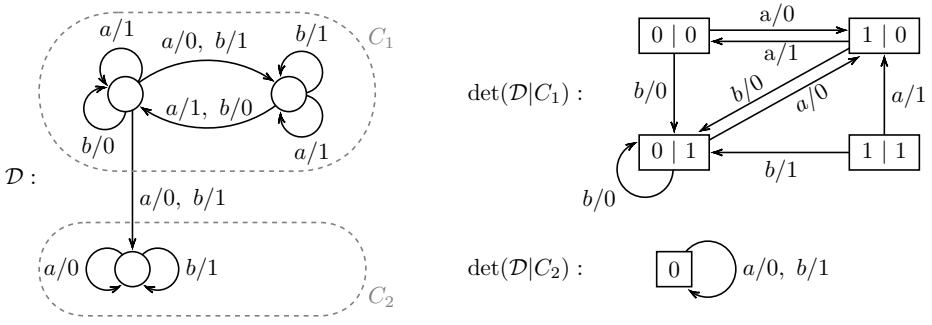


Fig. 1. A distance automaton with two SCCs and its determinized sub-automata

The asymptotic cost. We give an effective characterization of the asymptotic cost $\mathbf{A}(\mathcal{D})$ of a distance automaton $\mathcal{D}$ satisfying the shortcut property:

$$\mathbf{A}(\mathcal{D}) \;=^{\text{def}}\; \lim_{n \to \infty} \sup \left\{ \tfrac{\mathcal{D}(w)}{|w|} \;:\; w \in \Sigma^*,\ |w| \geq n \right\}.$$

The characterization will imply that the above value is rational and computable from $\mathcal{D}$. Before turning to the characterization, we remark that computability of asymptotic costs does not hold for arbitrary distance automata:

Proposition 3. *The problem of deciding, given an arbitrary distance automaton $\mathcal{D}$, whether or not $\mathbf{A}(\mathcal{D}) \leq \frac{1}{2}$ is undecidable.*

We use the undecidability of the $\frac{1}{2}$*-threshold problem* for normalized costs induced by distance automata [5], which consists of deciding, given a distance automaton $\mathcal{D}$, whether $\frac{\mathcal{D}(w)}{|w|} \leq \frac{1}{2}$ holds for all words $w \in \Sigma^*$. The reduction is done by transforming a given distance automaton $\mathcal{D}$ into a new distance automaton $\mathcal{D}'$ such that $\mathbf{A}(\mathcal{D}') = \sup \left\{ \frac{\mathcal{D}(w)}{|w|} \;:\; w \in \Sigma^* \right\}$.

Next we explain how the shortcut property helps in computing the asymptotic cost. One can show that the problem of computing $\mathbf{A}(\mathcal{D})$ for a distance automaton $\mathcal{D}$ that is *deterministic* is reducible to the problem of computing normalized costs of simple cycles. Formally, a *simple cycle* is any run of $\det(\mathcal{D})$ that is a cycle (i.e., that starts and ends in the same state) but that does not contain proper sub-cycles. It is then easy to show that for a deterministic distance automaton $\mathcal{D}$, $\mathbf{A}(\mathcal{D})$ coincides with the maximum of $\frac{\text{cost}(L)}{|L|}$ among all simple cycles L of $\mathcal{D}$, where $\text{cost}(L)$ denotes the cost of the simple cycle L. Thus by Proposition 2, calculation with simple cycles suffices to compute the asymptotic cost of any distance automaton satisfying the shortcut property and having a *single* SCC.

We consider now the more general case of a distance automaton $\mathcal{D}$ satisfying the shortcut property and having many SCCs, say $C_1, \ldots, C_k$. The situation in this case is slightly more complicated, as $\mathbf{A}(\mathcal{D})$ cannot be expressed as a function of $\mathbf{A}(\mathcal{D}|C_1), \ldots, \mathbf{A}(\mathcal{D}|C_k)$. We define $\bar{\mathcal{D}}$ as the deterministic *multi-distance* automaton obtained from the synchronous product of $\det(\mathcal{D}|C_1)$, …, $\det(\mathcal{D}|C_k)$ and we denote by $L_1, \ldots, L_m$ the simple cycles of $\bar{\mathcal{D}}$. Moreover, given $1 \leq i \leq m$

and $1 \leq j \leq k$, we denote by $\mathrm{cost}_j(L_i)$ the cost of the projection of the simple cycle L_i into the j-th component of $\bar{\mathcal{D}}$. Assuming that $\mathcal{D}$ is *trim*, namely, all its states are reachable from some states in $\mathcal{D}om(I)$ and they can reach some states in $\mathcal{D}om(F)$, we can characterize the asymptotic cost of $\mathcal{D}$ as follows:

Theorem 1. *For every distance automaton $\mathcal{D}$ satisfying the shortcut property,*

$$\mathbf{A}(\mathcal{D}) = \underbrace{\max_{\alpha_1,\ldots,\alpha_m \geq 0} \min_{1 \leq j \leq k} \frac{\sum_{1 \leq i \leq m} \alpha_i \cdot \mathrm{cost}_j(L_i)}{\sum_{1 \leq i \leq m} \alpha_i \cdot |L_i|}}_{\mathbf{E}(\mathcal{D})}. \tag{1}$$

The idea underlying the above characterization is that the asymptotic cost $\mathbf{A}(\mathcal{D})$ is achieved by repetitions of simple cycles in $\bar{\mathcal{D}}$. Indeed, the parameters $\alpha_1, \ldots, \alpha_m$ represent a correlation between the numbers of repetitions of the various simple cycles, and the index j represents the SCC of $\mathcal{D}$ that optimizes the normalized cost of these repetitions. The proof will consist of establishing two inequalities. In one case, we argue that all words can be approximated in cost by repetitions of simple cycles, and that the cost of editing these words is at most the cost of a "homogeneous strategy" that edits all cycles in the same component of $\mathcal{D}$. For the other inequality, we present a large family of words for which the best strategy is nearly homogeneous. The words will consist of nested repetitions of simple cycles in such a way that any edit strategy stabilizes by editing in the same component.

Example 5. Consider again the distance automaton $\mathcal{D}$ of Figure 1, with the two SCCs C_1 and C_2. The determinized sub-automaton $\det(\mathcal{D}|C_1)$ has four different simple cycles: one spelling aa with cost 1, one spelling ab with cost 0, one spelling b with cost 0, and one spelling aba with cost 1. Similarly, the determinized sub-automaton $\det(\mathcal{D}|C_2)$ has two simple cycles: one spelling a with cost 0, and the other spelling b with cost 1. Hence $(aa)^n$ is a family of words achieving a worst-case asymptotic cost of $\lim \frac{n}{2n} = \frac{1}{2}$ for the sub-automaton $\mathcal{D}|C_1$, and b^n is a family of words achieving a worst-case asymptotic cost of $\lim \frac{n}{n} = 1$ for the sub-automaton $\mathcal{D}|C_2$. However, a^{2n} is not a worst-case for $\mathcal{D}|C_2$ (as it can be repaired with asymptotic cost 0) and, symmetrically, b^n is not a worst-case for $\mathcal{D}|C_1$. This means that the worst-case asymptotic cost for $\mathcal{D}$ is achieved by a suitable combination of both families, namely, $(aab)^n$. This gives the asymptotic cost $\mathbf{A}(\mathcal{D}) = \lim \frac{n}{3n} = \frac{1}{3}$.

We make a few remarks related to the effectiveness of the characterization. First of all, we observe that the right handside term $\mathbf{E}(\mathcal{D})$ of Equation (1) can be rewritten as the following instance of a linear programming problem:

$$\texttt{maximize} \quad y \quad \texttt{subject to} \quad \begin{array}{ll} \sum_{1 \leq i \leq m} c_{i,j} \cdot x_i \geq y & \forall 1 \leq j \leq k \\ \sum_{1 \leq i \leq m} x_i \leq 1, \quad x_i \geq 0 & \forall 1 \leq i \leq k. \end{array}$$

where, for every $1 \leq i \leq m$ and every $1 \leq j \leq k$, $c_{i,j} = \frac{\mathrm{cost}_j(L_i)}{|L_i|}$. Intuitively, the variables $x_1, \ldots, x_m$ represent the values $\alpha_1 \cdot |L_1|$, …, $\alpha_m \cdot |L_m|$ normalized

in such a way that they sum up to 1, and the variable y represents an under-approximation of the value $\mathbf{E}(\mathcal{D})$. It is also known [6] that the optimal choices for the parameters $x_1, ..., x_m, y$ can be found at the 'corners' of the $(m+1)$-dimensional polyhedron that results from the intersection of the finitely many half-spaces defined by the above linear inequalities. This explains why we put $\max_{\alpha_1,...,\alpha_m \geq 0}$ instead of $\sup_{\alpha_1,...,\alpha_m \geq 0}$ in Equation (1). Moreover, it also implies that the asymptotic cost $\mathbf{A}(\mathcal{D})$ is a rational number.

Regarding the complexity of the problem of computing $\mathbf{A}(\mathcal{D})$, we observe that (i) the size $|\bar{\mathcal{D}}|$ of the multi-distance automaton $\bar{\mathcal{D}}$ is exponential in $|\mathcal{D}|$, (ii) each simple cycle L_i has length at most linear in $|\bar{\mathcal{D}}|$, (iii) the number m of simple cycles is exponential in $|\bar{\mathcal{D}}|$, and (iv) each constant $c_{i,j} = \frac{\text{cost}_j(L_i)}{|L_i|}$ can be computed in time polynomial in $|\bar{\mathcal{D}}|$ and $|L_i|$. Overall, the problem of computing the asymptotic cost of $\mathcal{D}$ is reduced, in time doubly exponential, to an instance of a linear programming problem. The latter problem is known to be in PTIME [3], which proves that $\mathbf{A}(\mathcal{D})$ can be computed in doubly exponential time.

From the cost of distance automata to the cost of editing. Theorem 1, together with Proposition 1 and Lemma 1, gives a way of computing the asymptotic cost $\mathbf{A}(\Sigma^*, \mathcal{T})$ of editing arbitrary words in Σ^* to words in $\mathscr{L}(\mathcal{T})$. Here we show how to generalize to our original problem, which involves the presence of both a restriction and a target language. We first modify the definition of asymptotic cost for a distance automaton to include the presence of a restriction language $\mathscr{L}(\mathcal{R})$ recognized by a DFA $\mathcal{R}$:

$$\mathbf{A}(\mathcal{R}, \mathcal{D}) \ =^{\text{def}} \ \lim_{n \to \infty} \sup \left\{ \frac{\mathcal{D}(w)}{|w|} \ : \ w \in \mathscr{L}(\mathcal{R}), \ |w| \geq n \right\}.$$

Given a DFA $\mathcal{R}$ and a distance automaton $\mathcal{D}$ satisfying the shortcut property, we denote by $\mathcal{D}ag(\mathcal{R})$ (resp., $\mathcal{D}ag(\mathcal{D})$) the directed acyclic graph of the SCCs of $\mathcal{R}$ (resp., $\mathcal{D}$). The paths in $\mathcal{D}ag(\mathcal{R})$ (resp., $\mathcal{D}ag(\mathcal{D})$) are the sequences of SCCs of the form $\pi = C_1 \dots C_h$, where each SCC C_{l+1} is reachable from the previous SCC C_l. Given a SCC B of $\mathcal{R}$, we denote by $L_{B,1}, \dots, L_{B,m_B}$ the simple cycles of the automaton $\bar{\mathcal{D}} \times (\mathcal{R}|B) = \det(\mathcal{D}|C_1) \times \dots \times \det(\mathcal{D}|C_k) \times (\mathcal{R}|C)$, where $C_1, \dots, C_k$ are the SCCs of $\mathcal{D}$ and $\mathcal{R}|B$ is the sub-automaton obtained from $\mathcal{R}$ by restricting the set of states to B (it does not matter which state is chosen to be initial in $\mathcal{R}|B$). Finally, given a simple cycle $L_{B,i}$ of $\bar{\mathcal{D}} \times (\mathcal{R}|B)$ and a SCC C of $\mathcal{D}$, we denote by $\text{cost}_C(L_{B,i})$ the cost of a the projection of $L_{B,i}$ into the component C of $\bar{\mathcal{D}} \times (\mathcal{R}|B)$. The generalized characterization result is as follows:

Theorem 2. *For every (trim) DFA $\mathcal{R}$ and every distance automaton $\mathcal{D}$ satisfying the shortcut property,*

$$\mathbf{A}(\mathcal{R}, \mathcal{D}) \ = \ \max_{\substack{\tau = B_1 \dots B_h \in \mathcal{D}ag(\mathcal{R}) \\ \alpha_{1,1},\dots,\alpha_{1,m_1} \geq 0 \\ \dots \\ \alpha_{h,1},\dots,\alpha_{h,m_h} \geq 0}} \ \min_{\pi = C_1 \dots C_h \in \mathcal{D}ag(\mathcal{D})} \frac{\sum_{1 \leq l \leq h} \sum_{1 \leq i \leq m_l} \alpha_{l,i} \cdot \text{cost}_{C_l}(L_{B_l,i})}{\sum_{1 \leq l \leq h} \sum_{1 \leq i \leq m_l} \alpha_{l,i} \cdot |L_{B_l,i}|}$$

Using arguments similar to the complexity analysis in the unrestricted case, we obtain that the asymptotic cost $\mathbf{A}(\mathcal{R}, \mathcal{T})$ $(= \mathbf{A}(\mathcal{R}, \mathcal{D}^{\text{edit}}_{\mathcal{T}}))$ for two DFA $\mathcal{R}$ and $\mathcal{T}$ is computable in 2EXPTIME.

4 Asymptotic Cost in the Streaming Case

Here we characterize the asymptotic cost in the streaming setting in terms of the value of a mean-payoff game [2]. A *mean-payoff game* is an infinite, turn-based game played over an arena $\mathcal{M} = (V, E, v_0)$, where V is the union of two disjoint finite sets of vertices, V_{Adam} (owned by player Adam) and V_{Eve} (owned by player Eve), $E \subseteq V \times \mathbb{N} \times V$ is a finite set of weighted edges, and $v_0 \in V$ is an initial vertex. The game starts at v_0 and, at each round, the player who owns the current vertex v moves along an edge $(v, c, v') \in E$. The reward for Adam (resp., Eve) in an infinite play $\pi = (v_0, c_1, v_1)\ (v_1, c_2, v_2)\ldots$ is given by the value ν^{π}_{Adam} (resp., $-\nu^{\pi}_{\text{Eve}}$), where

$$\nu^{\pi}_{\text{Adam}} =^{\text{def}} \liminf_{n \to \infty} \frac{\sum_{i=1}^{n} c_i}{n} \qquad \nu^{\pi}_{\text{Eve}} =^{\text{def}} \limsup_{n \to \infty} \frac{\sum_{i=1}^{n} c_i}{n}$$

Intuitively, Adam wants to maximize ν^{π}_{Adam} and Eve wants to minimize ν^{π}_{Eve}.

It is known from [2] that mean-payoff games are positionally determined, namely, to each mean-payoff game corresponds a value ν such that Adam (resp., Eve) has a positional strategy that guarantees $\nu^{\pi}_{\text{Adam}} \geq \nu$ (resp., $\nu^{\pi}_{\text{Eve}} \leq \nu$) for all plays π that respect his (resp., her) strategy.

Let $\mathcal{R} = (\Sigma, Q, \delta, q_0, F)$ and $\mathcal{T} = (\Delta, Q', \delta', q'_0, F')$ be two trim DFA. To compute the streaming asymptotic cost $\mathbf{SA}(\mathcal{R}, \mathcal{T})$, we construct the arena $\mathcal{M}^{\text{edit}}_{\mathcal{R},\mathcal{T}}$, where Adam's vertices are pairs of the form (q, q'), with $q \in Q$ and $q' \in Q'$, and Eve's vertices are pairs of the form (q, q', a), with $q \in Q$, $q' \in Q'$, and $a \in \Sigma$. The edges of the arena are triples of the form $\big((q, q'), 0, (p, q', a)\big)$, where $p = \delta(q, a)$, or of the form $\big((q, q', a), c, (q, p')\big)$, where $c = \min\{\text{edit-dist}(a, v) : v \in \mathscr{L}(\mathcal{T}_{q',p'})\}$. The initial vertex of the arena is the pair (q_0, q'_0) (so Adam moves first). Observe that the final states of $\mathcal{R}$ and $\mathcal{T}$ do not play any relevant role in this definition: this is because $\mathcal{R}$ and $\mathcal{T}$ are assumed to be trim and the costs of moving from non-final states to final states are irrelevant for the asymptotic behaviour. Furthermore, note that the game alternates between Adam and Eve, and only the second player can incur positive costs.

Below, we show that the value of the mean-payoff game over $\mathcal{M}^{\text{edit}}_{\mathcal{R},\mathcal{T}}$, multiplied by 2, coincides with the asymptotic cost in the streaming setting.

Theorem 3. *Given two DFA $\mathcal{R}$ and $\mathcal{T}$, we have $\mathbf{SA}(\mathcal{R}, \mathcal{T}) = 2 \cdot \nu$, where ν is the value of the mean-payoff game over $\mathcal{M}^{\text{edit}}_{\mathcal{R},\mathcal{T}}$. Moreover, $\mathbf{SA}(\mathcal{R}, \mathcal{T})$ is rational, it can be computed in polynomial time, and it is achieved by a single streaming edit strategy for $\mathscr{L}(\mathcal{R})$ and $\mathscr{L}(\mathcal{T})$ – which can also be computed in PTIME.*

Even if it seems natural that the value of the mean-payoff game over $\mathcal{M}^{\text{edit}}_{\mathcal{R},\mathcal{T}}$ determines the asymptotic cost $\mathbf{SA}(\mathcal{R}, \mathcal{T})$, we remark that the proof of the above theorem is not trivial. Indeed, the mean-payoff game corresponds directly to a version of the streaming edit problem where the input to the edit strategy is a sequence of prefixes of a single infinite word spelled by a run of $\mathcal{R}$. The core of the proof is to show a correspondence between this infinitary version of the streaming edit problem and the original problem as stated in Section 2. This is done by showing that $(\star)$ for the optimal strategy of Eve $\mathcal{S}$ in the mean-payoff

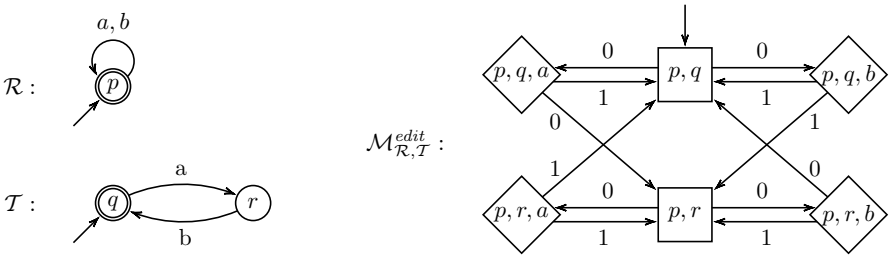


Fig. 2. Two DFA and the arena for the associated mean-payoff game

game, one can construct a streaming edit processor $\mathcal{S}'$ of $\mathcal{R}$ into $\mathcal{T}$ such that $\frac{\mathbf{A}(\mathcal{R},\mathcal{S}',\mathcal{T})}{2}$ does not exceed the reward of $\mathcal{S}$. $\mathcal{S}'$ just mimics $\mathcal{S}$ until the string terminates, at which point it performs additional insertions to get to a final state. For the other direction we take any streaming edit processor $\mathcal{S}'$ of $\mathcal{R}$ into $\mathcal{T}$ with value $\mathbf{A}(\mathcal{R},\mathcal{S}',\mathcal{T})$ and show that no strategy $\mathcal{S}$ for Adam can guarantee a reward of more than $\frac{\mathbf{A}(\mathcal{R},\mathcal{S}',\mathcal{T})}{2}$. By the result from [2] mentioned above, this shows that Eve can guarantee a reward of at least this amount. The limit on Adam's ability is shown by combating his strategy $\mathcal{S}$ using the edit processor $\mathcal{S}'$. Putting these two directions together, we see that the optimal streaming edit processor is produced by first computing Eve's optimal strategy, then applying the transformation $(\star)$ described above: we will argue below that this is a PTIME procedure.

We recall that the problem of deciding whether the value of an arbitrary mean-payoff game $\mathcal{M} = (V, E, v_0)$ is below a certain threshold is in coNP $\cap$ NP; much recent work has focused on improving the exponential bounds on deterministic algorithms; for example, it can be done in $\mathcal{O}(|V|^2 \cdot |E| \cdot c_{\max})$, where $c_{\max}$ is the maximum weight of an edge of $\mathcal{M}$ [12]. Even though the parameter $c_{\max}$ is exponential when the weights are represented in binary notation, when restricting to arenas of the form $\mathcal{M}^{\text{edit}}_{\mathcal{R},\mathcal{T}}$, this value never exceeds the total number of states of the DFA $\mathcal{T}$. This gives the polynomial bound on the complexity of the problem of computing the value of the mean-payoff game over $\mathcal{M}^{\text{edit}}_{\mathcal{R},\mathcal{T}}$, and the PTIME bound in Theorem 3 follows.

Example 6. Consider the restriction and target language $R : (a + b)^*$ and $T : (ab)^*$ which automata and respectively mean-payoff arena $\mathcal{M}^{\text{edit}}_{\mathcal{R},\mathcal{T}}$ are shown in Figure 2. Here, diamond nodes are owned by Eve and square nodes are owned by Adam. One can easily see that an optimal positional strategy for Adam is to play $(p, q) \longrightarrow (p, q, b)$ and $(p, r) \longrightarrow (p, r, a)$. With this strategy we get that for every Eve's strategy the value ν of the mean-payoff game over $\mathcal{M}^{\text{edit}}_{\mathcal{R},\mathcal{T}}$ is equal to $\frac{1}{2}$ and then $\mathbf{SA}(R, T) = 1$. This value definitely contrasts with the non-streaming asymptotic cost between R and T which is equal to $\frac{1}{2}$.

There is a natural generalization of the above theorem for computing the asymptotic cost of streaming edits with k-lookahead: it is indeed sufficient to modify the

definition of the arena $\mathcal{M}^{\text{edit}}_{\mathcal{R},\mathcal{T}}$ in such a way that Adam plays $(k+1)$-character windows. Note that this requires extending the set of vertices of $\mathcal{M}^{\text{edit}}_{\mathcal{R},\mathcal{T}}$ from $\big(Q \times Q'\big) \cup \big(Q \times Q' \times \Sigma\big)$ to $\big(Q \times Q' \times (\Sigma \cup \{\bot\})^k\big) \cup \big(Q \times Q' \times \Sigma \times (\Sigma \cup \{\bot\})^k\big)$.

5 Conclusions

We have addressed the problem of computing the asymptotic cost between regular languages in the non-streaming and streaming settings. It is surprising that the asymptotic cost in both settings is rational and computable. In the streaming setting this gives us optimal online algorithms for editing one language into another, which are quite distinct from traditional edit distance algorithms based on dynamic programming. We leave as an open problem whether the algorithms for computing asymptotic cost in the nonstreaming setting are optimal.

Acknowledgments. We thank the anonymous referees, Thomas Colcombet, and Slawek Staworko for many helpful comments. The authors were supported by EPSRC (UK) grant EP/G004021/1.

References

[1] Benedikt, M., Puppis, G., Riveros, C.: Regular repair of specifications. In: LICS (2011)
[2] Ehrenfeucht, A., Mycielski, J.: Positional strategies for mean payoff games. J. of Game Theory 8, 109–113 (1979)
[3] Karmarkar, N.: A new polynomial-time algorithm for linear programming. In: STOC, pp. 302–311 (1984)
[4] Konstantinidis, S.: Computing the edit distance of a regular language. Inf. and Comp. 205(9), 1307–1316 (2007)
[5] Krob, D.: The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 101–112. Springer, Heidelberg (1992)
[6] Matouek, J., Gärtner, B.: Understanding and Using Linear Programming (2006)
[7] Mohri, M.: Finite-state transducers in language and speech processing. J. of Comp. Ling. 23(2), 269–311 (1997)
[8] Mohri, M.: Edit-distance of weighted automata: general definitions and algorithms. J. of Found. of Comp. Sc. 14(6), 957–982 (2003)
[9] Simon, I.: Recognizable sets with multiplicities in the tropical semiring. In: Koubek, V., Janiga, L., Chytil, M.P. (eds.) MFCS 1988. LNCS, vol. 324, pp. 107–120. Springer, Heidelberg (1988)
[10] Wagner, R.A.: Order-n correction for regular languages. CACM 17(5), 265–268 (1974)
[11] Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. JACM 21(1), 168–173 (1974)
[12] Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theor. Comput. Sci. 158, 343–359 (1996)

Emptiness and Universality Problems in Timed Automata with Positive Frequency*

Nathalie Bertrand[1], Patricia Bouyer[2], Thomas Brihaye[3], and Amélie Stainer[1]

[1] INRIA Rennes, France
[2] LSV - CNRS & ENS Cachan, France
[3] Université de Mons, Belgium

Abstract. The languages of infinite timed words accepted by timed automata are traditionally defined using Büchi-like conditions. These acceptance conditions focus on the set of locations visited infinitely often along a run, but completely ignore quantitative timing aspects. In this paper we propose a natural quantitative semantics for timed automata based on the so-called frequency, which measures the proportion of time spent in the accepting locations. We study various properties of timed languages accepted with positive frequency, and in particular the emptiness and universality problems.

1 Introduction

The model of timed automata, introduced by Alur and Dill in the 90's [2] is commonly used to represent real-time systems. Timed automata consist of an extension of finite automata with continuous variables, called clocks, that evolve synchronously with time, and can be tested and reset along an execution. Despite their uncountable state space, checking reachability, and more generally ω-regular properties, is decidable *via* the construction of a finite abstraction, the so-called region automaton. This fundamental result made timed automata very popular in the formal methods community, and lots of work has been done towards their verification, including the development of dedicated tools like Kronos or Uppaal.

More recently a huge effort has been made for modelling quantitative aspects encompassing timing constraints, such as costs [3,6] or probabilities [11,5]. It is now possible to express and check properties such as: "the minimal cost to reach a given state is smaller than 3", or "the probability to visit infinitely often a given location is greater than 1/2". As a consequence, from qualitative verification, the emphasis is now put on quantitative verification of timed automata.

In this paper we propose a quantitative semantics for timed automata based on the proportion of time spent in critical states (called the *frequency*). Contrary

* This work has been partly supported by the ESF project GASICS, the ANR project ANR-2010-BLAN-0317, the Tournesol Hubert Curien partnership STP, the ARC project AUWB-2010–10/15-UMONS-3, the FRFC project 2.4515.11 and a grant from the National Bank of Belgium.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 246–257, 2011.

to probabilities or volume [4] that give a value to sets of behaviours of a timed automaton (or a subset thereof), the frequency assigns a real value (in $[0, 1]$) to each execution of the system. It can thus be used in a language-theoretic approach to define quantitative languages associated with a timed automaton, or boolean languages based on quantitative criteria *e.g.*, one can consider the set of timed words for which there is an execution of frequency greater than a threshold λ.

Similar notions were studied in the context of untimed systems. For finite automata, mean-payoff conditions have been investigated [10,1,9]: with each run is associated the limit average of weights encountered along the execution. Our notion of frequency extends mean-payoff conditions to timed systems by assigning to an execution the limit average of time spent in some distinguished locations. It can also be seen as a timed version of the asymptotic frequency considered in quantitative fairness games [7]. Concerning probabilistic models, a similar notion was introduced in constrained probabilistic Büchi automata yielding the decidability of the emptiness problem under the probable semantics [14]. Last, the work closest to ours deals with double-priced timed automata [8], where the aim is to synthesize schedulers which optimize on-the-long-term the reward of a system.

Adding other quantitative aspects to timed automata comes often with a cost (in terms of decidability and complexity), and it is often required to restrict the timing behaviours of the system to get some computability results, see for instance [13]. The tradeoff is then to restrict to single-clock timed automata. Beyond introducing the concept of frequency, which we believe very natural, the main contributions of this paper are the following. First of all, using a refinement of the region automaton abstraction, we show how to compute the infimum and supremum values of frequencies in a given single-clock timed automaton, as well as a way to decide whether these bounds are realizable (*i.e.*, whether they are minimum and maximum respectively). The computation of these bounds together with their realizability can be used to decide the emptiness problem for languages defined by a threshold on the frequency. Moreover, in the restricted case of deterministic timed automata, it allows to decide the universality problem for these languages. Last but not least we discuss the universality problem for frequency-languages. Even under our restriction to one-clock timed automata, this problem is non-primitive recursive, and we provide a decision algorithm in the case of Zeno words when the threshold is 0. Our restriction to single-clock timed automata is crucial since at several points the techniques employed do not extend to two clocks or more. In particular, the universality problem becomes undecidable for timed automata with several clocks.

2 Definitions and Preliminaries

In this section, we recall the model of timed automata, introduce the concept of frequency, and show how those can be used to define timed languages. We then compare our semantics to the standard semantics based on Büchi acceptance.

2.1 Timed Automata and Frequencies

We start with notations and useful definitions concerning timed automata [2].

Given X a finite set of clocks, a *(clock) valuation* is a mapping $v : X \to \mathbb{R}_+$. We write $\mathbb{R}_+^X$ for the set of valuations. We note $\overline{0}$ the valuation that assigns 0 to all clocks. If v is a valuation over X and $t \in \mathbb{R}_+$, then $v+t$ denotes the valuation which assigns to every clock $x \in X$ the value $v(x) + t$. For $X' \subseteq X$ we write $v_{[X' \leftarrow 0]}$ for the valuation equal to v on $X \setminus X'$ and to $\overline{0}$ on X'.

A *guard* over X is a finite conjunction of constraints of the form $x \sim c$ where $x \in X$, $c \in \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$. We denote by $G(X)$ the set of guards over X. Given g a guard and v a valuation, we write $v \models g$ if v satisfies g (defined in a natural way).

Definition 1. *A* timed automaton *is a tuple $\mathcal{A} = (L, L_0, F, \Sigma, X, E)$ such that: L is a finite set of locations, $L_0 \subseteq L$ is the set of initial locations, $F \subseteq L$ is the set of accepting locations, Σ is a finite alphabet, X is a finite set of clocks and $E \subseteq L \times G(X) \times \Sigma \times 2^X \times L$ is a finite set of edges.*

The semantics of a timed automaton $\mathcal{A}$ is given as a timed transition system $\mathcal{T}_\mathcal{A} = (S, S_0, S_F, (\mathbb{R}_+ \times \Sigma), \to)$ with set of states $S = L \times \mathbb{R}_+^X$, initial states $S_0 = \{(\ell_0, \overline{0}) \mid \ell_0 \in L_0\}$, final states $S_F = F \times \mathbb{R}_+^X$ and transition relation $\to \subseteq S \times (\mathbb{R}_+ \times \Sigma) \times S$, composed of moves of the form $(\ell, v) \xrightarrow{\tau, a} (\ell', v')$ with $\tau > 0$ whenever there exists an edge $(\ell, g, a, X', \ell') \in E$ such that $v + \tau \models g$ and $v' = (v + \tau)_{[X' \leftarrow 0]}$.

A *run* ϱ of $\mathcal{A}$ is an infinite sequence of moves starting in some $s_0 \in S_0$, *i.e.*, $\varrho = s_0 \xrightarrow{\tau_0, a_0} s_1 \cdots \xrightarrow{\tau_k, a_k} s_{k+1} \cdots$. A *timed word* over Σ is an element $(t_i, a_i)_{i \in \mathbb{N}}$ of $(\mathbb{R}_+ \times \Sigma)^\omega$ such that $(t_i)_{i \in \mathbb{N}}$ is increasing. The timed word is said to be *Zeno* if the sequence $(t_i)_{i \in \mathbb{N}}$ is bounded from above. The timed word associated with ϱ is $w = (t_0, a_0) \dots (t_k, a_k) \dots$ where $t_i = \sum_{j=0}^{i} \tau_j$ for every i. A timed automaton $\mathcal{A}$ is *deterministic* whenever, given two edges $(\ell, g_1, a, X_1', \ell')$ and $(\ell, g_2, a, X_2', \ell')$ in E, $g_1 \wedge g_2$ cannot be satisfied. In this case, for every timed word w, there is at most one run reading w. An example of a (deterministic) timed automaton is given in Fig. 1. As a convention locations in F will be depicted in grey.

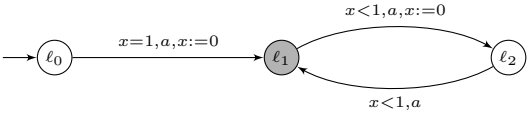


Fig. 1. Example of a timed automaton $\mathcal{A}$ with $L_0 = \{\ell_0\}$ and $F = \{\ell_1\}$

Definition 2. *Given $\mathcal{A} = (L, L_0, F, \Sigma, X, E)$ a timed automaton and a run $\varrho = (\ell_0, v_0) \xrightarrow{\tau_0, a_0} (\ell_1, v_1) \xrightarrow{\tau_1, a_1} (\ell_2, v_2) \cdots$ of $\mathcal{A}$, the* frequency *of F along ϱ, denoted $\mathsf{freq}_\mathcal{A}(\varrho)$, is defined as $\limsup_{n \to \infty} (\sum_{i \leq n \mid \ell_i \in F} \tau_i) / (\sum_{i \leq n} \tau_i)$.*

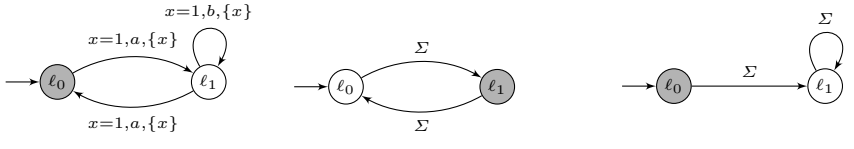
Note that the choice of lim sup is arbitrary, and the choice of lim inf would be as relevant. Furthermore notice that the limit may not exist in general.

A timed word w is said *accepted with positive frequency* by $\mathcal{A}$ if there exists a run ϱ which reads w and such that $\mathsf{freq}_{\mathcal{A}}(\varrho)$ is positive. The *positive-frequency language* of $\mathcal{A}$ is the set of timed words that are accepted with positive frequency by $\mathcal{A}$. Note that we could define more generally languages where the frequency of each word should be larger than some threshold λ, but even though some of our results apply to this more general framework we prefer focusing on languages with positive frequency.

Example 3. We illustrate the notion of frequency on runs of the deterministic timed automaton $\mathcal{A}$ of Fig. 1. First, the only run in $\mathcal{A}$ 'reading' the word $(1,a).((\frac{1}{3},a).(\frac{1}{3},a))^*$ has frequency $\frac{1}{2}$ because the sequence $\frac{n/3}{1+(2n)/3}$ converges to $\frac{1}{2}$. Second, the Zeno run reading $(1,a).(((\frac{1}{2^k},a).(\frac{1}{2^k},a))^k)_{k\geq 1}$ in $\mathcal{A}$ has frequency $\frac{1}{3}$ since the sequence $\frac{\sum_{k\geq 1} 1/2^k}{1+\sum_{k\geq 1} 1/2^{k-1}}$ converges to $\frac{1}{3}$. Finally, the run in $\mathcal{A}$ reading the word $(1,a).(((\frac{1}{2},a).(\frac{1}{4},a))^{2^{2k}}.((\frac{1}{4},a).(\frac{1}{2},a))^{2^{2k+1}})_{k\geq 1}$ has frequency $\frac{4}{9}$. Note that the sequence under consideration does not converge, but its lim sup is $\frac{4}{9}$.

2.2 A Brief Comparison with Usual Semantics

The usual semantics for timed automata considers a Büchi acceptance condition. We naturally explore differences between this usual semantics, and the one we introduced based on positive frequency. The expressiveness of timed automata under those acceptance conditions is not comparable, as witnessed by the automaton represented in Fig. 2(a): on the one hand, its positive-frequency language is not timed-regular (*i.e.* accepted by a timed automaton with a standard Büchi acceptance condition), and on the other hand, its Büchi language cannot be recognized by a timed automaton with a positive-frequency acceptance condition.



(a) Expressiveness (b) Universality (non-Zeno) (c) Universality (Zeno)

Fig. 2. Automata for the comparison with the usual semantics

The contribution of this paper is to study properties of the positive-frequency languages. We will show that we can get very fine information on the set of frequencies of runs in *single-clock* timed automata, which implies the decidability of the emptiness problem for positive-frequency languages. We also show that our technics do not extend to multi-clock timed automata.

We will also consider the universality problem and variants thereof (restriction to Zeno or non-Zeno timed words). On the one hand, clearly enough, a (non-Zeno)-universal timed automaton with a positive-frequency acceptance condition

is (non-Zeno)-universal for the classical Büchi-acceptance. The timed automaton of Fig. 2(b) is a counterexample to the converse. On the other hand, a Zeno-universal timed automaton under the classical semantics is necessarily Zeno-universal under the positive-frequency acceptance condition, but the automaton depicted in Fig. 2(c) shows that the converse does not hold.

3 Set of Frequencies of Runs in One-Clock Timed Automata

In this section, we give a precise description of the set of frequencies of runs in single-clock timed automata. To this aim, we use the *corner-point abstraction* [8], a refinement of the region abstraction, and exploit the links between frequencies in the timed automaton and ratios in its corner-point abstraction. We fix a single-clock timed automaton $\mathcal{A} = (L, L_0, F, \Sigma, \{x\}, E)$.

3.1 The Corner-Point Abstraction

Even though the corner-point abstraction can be defined for general timed automata [8], we focus on the case of single-clock timed automata.

If M is the largest constant appearing in the guards of $\mathcal{A}$, the usual *region abstraction* of $\mathcal{A}$ is the partition $Reg_{\mathcal{A}}$ of the set of valuations $\mathbb{R}_+$ made of the singletons $\{i\}$ for $0 \leq i \leq M$, the open intervals $(i, i+1)$ with $0 \leq i \leq M-1$ and the unbounded interval (M, ∞) represented by $\perp$. A piece of this partition is called a *region*. The corner-point abstraction refines the region abstraction by associating *corner-points* with regions. The singleton regions have a single corner-point represented by $\bullet$ whereas the open intervals $(i, i+1)$ have two corner-points $\bullet\!-$ (the left end-point of the interval) and $-\!\bullet$ (the right end-point of the interval). Finally, the region $\perp$ has a single corner-point denoted $\alpha_\perp$. We write (R, α) for the region R pointed by the corner α and $(R, \alpha) + 1$ denotes its *direct time successor* defined by:

$$(R,\alpha)+1 = \begin{cases} ((i,i+1), \bullet\!-) & \text{if } (R,\alpha) = (\{i\}, \bullet) \text{ with } i < M, \\ ((i,i+1), -\!\bullet) & \text{if } (R,\alpha) = ((i,i+1), \bullet\!-), \\ (\{i+1\}, \bullet) & \text{if } (R,\alpha) = ((i,i+1), -\!\bullet), \\ (\perp, \alpha_\perp) & \text{if } (R,\alpha) = (\{M\}, \bullet) \text{ or } (\perp, \alpha_\perp). \end{cases}$$

Using these notions, we define the corner-point abstraction as follows.

Definition 4. *The* (unweighted) corner-point abstraction *of $\mathcal{A}$ is the finite automaton $\mathcal{A}_{cp} = (L_{cp}, L_{0,cp}, F_{cp}, \Sigma_{cp}, E_{cp})$ where $L_{cp} = L \times Reg_{\mathcal{A}} \times \{\bullet, \bullet\!-, -\!\bullet, \alpha_\perp\}$ is the set of states, $L_{0,cp} = L_0 \times \{0\} \times \{\bullet\}$ is the set of initial states, $F_{cp} = F \times Reg_{\mathcal{A}} \times \{\bullet, \bullet\!-, -\!\bullet, \alpha_\perp\}$ is the set of accepting states, $\Sigma_{cp} = \Sigma \cup \{\varepsilon\}$, and $E_{cp} \subseteq L_{cp} \times \Sigma_{cp} \times L_{cp}$ is the finite set of edges defined as the union of discrete transitions and idling transitions:*

- discrete transitions: $(\ell, R, \alpha) \xrightarrow{a} (\ell', R', \alpha')$ *if α is a corner-point of R and there exists a transition $\ell \xrightarrow{g,a,X'} \ell'$ in $\mathcal{A}$, such that $R \subseteq g$ and $(R', \alpha') = (R, \alpha)$ if $X' = \emptyset$, otherwise $(R', \alpha') = (\{0\}, \bullet)$,*

– idling transitions: $(\ell, R, \alpha) \xrightarrow{\varepsilon} (\ell, R', \alpha')$ *if* α *(resp.* α'*) is a corner-point of* R *(resp.* R'*) and* $(R', \alpha') = (R, \alpha) + 1$.

We decorate this finite automaton with two weights for representing frequencies, one which we call the cost, and the other which we call the reward (by analogy with double-priced timed automata in [8]). The *(weighted) corner-point abstraction* $\mathcal{A}^F_{cp}$ is obtained from $\mathcal{A}_{cp}$ by labeling idling transitions in $\mathcal{A}_{cp}$ as follows: transitions $(\ell, R, \alpha) \xrightarrow{\varepsilon} (\ell, R, \alpha')$ with $(R, \alpha') = (R, \alpha) + 1$ ($\alpha' = \alpha + 1$ for short) are assigned cost 1 (resp. cost 0) and reward 1 if $\ell \in F$ (resp. $\ell \notin F$), and all other transitions are assigned both cost and reward 0. To illustrate this definition, the corner-point abstraction of the timed automaton in Fig. 1 is represented in Fig. 3.

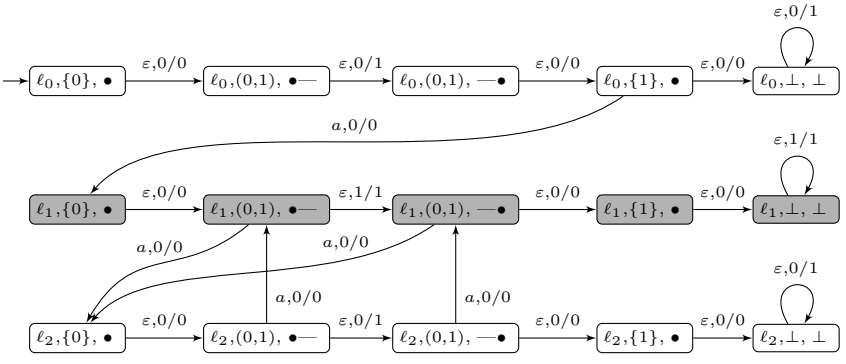


Fig. 3. The corner-point abstraction $\mathcal{A}^F_{cp}$ of $\mathcal{A}$ represented Fig. 1

There will be a correspondence between runs in $\mathcal{A}$ and runs in $\mathcal{A}_{cp}$. As time is increasing in $\mathcal{A}$ we forbid runs in $\mathcal{A}_{cp}$ where two actions have to be made in 0-delay (this is easy to do as there should be no sequence $\ldots \xrightarrow{\sigma} (\ell, R, \alpha) \xrightarrow{\sigma'} \ldots$, where both σ and σ' are actions and R is a punctual region).

Given π a run in $\mathcal{A}^F_{cp}$ the *ratio* of π, denoted $\mathsf{Rat}(\pi)$, is defined, provided it exists, as the lim sup of the ratio of accumulated costs divided by accumulated rewards for finite prefixes. Run π is said *reward-converging* (resp. *reward-diverging*) if the accumulated reward along π is bounded (resp. unbounded). Reward-converging runs in $\mathcal{A}^F_{cp}$ are meant to capture Zeno behaviours of $\mathcal{A}$.

Given ϱ a run in $\mathcal{A}$ we denote by $\mathsf{Proj}_{cp}(\varrho)$ the set of all runs in $\mathcal{A}^F_{cp}$ compatible with ϱ in the following sense. We assume $\varrho = (\ell_0, v_0) \xrightarrow{\tau_0, a_0} (\ell_1, v_1) \xrightarrow{\tau_1, a_1} \cdots$, where move $(\ell_i, v_i) \xrightarrow{\tau_i, a_i} (\ell_{i+1}, v_{i+1})$ comes from an edge e_i. A run[1] $\pi = (\ell_0, R^1_0, \alpha^1_0) \to (\ell_0, R^2_0, \alpha^2_0) \to \cdots \to (\ell_0, R^{k_0}_0, \alpha^{k_0}_0) \to (\ell_1, R^1_1, \alpha^1_1) \to \cdots \to (\ell_1, R^{k_1}_1, \alpha^{k_1}_1) \cdots$ of $\mathcal{A}^F_{cp}$ is in $\mathsf{Proj}_{cp}(\varrho)$ if for all indices $n \geq 0$:

– for all $i \leq k_n$, α^i_n is a corner-point of R^i_n,
– for all $i \leq k_n - 1$, $(R^{i+1}_n, \alpha^{i+1}_n) = (R^i_n, \alpha^i_n) + 1$,

[1] For simplicity, we omit here the transitions labels.

- $(R^1_{n+1}, \alpha^1_{n+1})$ is the successor pointed-region of $(R^{k_n}_n, \alpha^{k_n}_n)$ by transition e_n (that is $(R^1_{n+1}, \alpha^1_{n+1}) = (\{0\}, \bullet)$ if e_n resets the clock x and otherwise $(R^1_{n+1}, \alpha^1_{n+1}) = (R^{k_n}_n, \alpha^{k_n}_n)$),
- $v_n \in R^1_n$ and if $R^{k_n}_n \neq \bot$, $v_n + \tau_n \in R^{k_n}_n$,
- if $R^{k_n}_n = \bot$, the sum μ_n of the rewards since region $\{0\}$ has been visited for the last time has to be equal to $\lfloor v_n + \tau_n \rfloor$ or $\lceil v_n + \tau_n \rceil$.[2] Note that μ_n can be seen as the abstraction of the valuation v_n.

Remark 5. As defined above, the size of $\mathcal{A}^F_{cp}$ is exponential in the size of $\mathcal{A}$ because the number of regions is $2M$ (which is exponential in the binary encoding of M). We could actually take a rougher version of the regions [12], where only constants appearing in $\mathcal{A}$ should take part in the region partition. This partition, specific to single-clock timed automata is only polynomial in the size of $\mathcal{A}$. We choose to simplify the presentation by considering the standard unit intervals.

We will now see that the corner-point abstraction is a useful tool to deduce properties of the set of frequencies of runs in the original timed automata.

3.2 From $\mathcal{A}$ to $\mathcal{A}^F_{cp}$, and *Vice-Versa*

We first show that given a run ϱ of $\mathcal{A}$, there exists a run in $\mathsf{Proj}_{cp}(\varrho)$, whose ratio is smaller (resp. larger) than the frequency of ϱ.

Lemma 6 (From $\mathcal{A}$ to $\mathcal{A}^F_{cp}$). *For every run ϱ in $\mathcal{A}$, there exist π and π' in $\mathcal{A}^F_{cp}$ that can effectively be built and belong to $\mathsf{Proj}_{cp}(\varrho)$ such that:*

$$\mathsf{Rat}(\pi) \leq \mathsf{freq}_{\mathcal{A}}(\varrho) \leq \mathsf{Rat}(\pi').$$

Run π (resp. π') minimizes (resp. maximizes) the ratio among runs in $\mathsf{Proj}_{cp}(\varrho)$.

Such two runs of $\mathcal{A}^F_{cp}$ can be effectively built from ϱ, through the so-called *contraction* (resp. *dilatation*) operations. Intuitively it consists in minimizing (resp. maximizing) the time elapsed in F-locations.

Note that the notion of contraction cannot be adapted to the case of timed automata with several clocks, as illustrated by the timed automaton in Fig. 4. Consider indeed the run alternating delays $(\frac{1}{2} + \frac{1}{n})$ and $1 - (\frac{1}{2} + \frac{1}{n})$ for $n \in \mathbb{N}$, and switching between the left-most cycle $(\ell_1 - \ell_2 - \ell_1)$ and the right-most cycle $(\ell_3 - \ell_4 - \ell_3)$ following the rules: in round k, take 2^{2k} times the cycle $\ell_1 - \ell_2 - \ell_1$, then switch to ℓ_3 and take 2^{2k+1} times the cycle $\ell_3 - \ell_4 - \ell_3$ and return back to ℓ_1 and continue with round $k+1$. This run cannot have any contraction since its frequency is $\frac{1}{2}$, whereas all its projections in the corner-point abstraction have ratio $\frac{2}{3}$, the lim sup of a non-converging sequence. This strange behavior is due to the fact that the delays in ℓ_1 and ℓ_3 need to be smaller and smaller, and this converging phenomenon requires at least two clocks.

We now want to know when and how runs in $\mathcal{A}^F_{cp}$ can be lifted to $\mathcal{A}$. To that aim we distinguish between reward-diverging and reward-converging runs.

[2] Roughly, in the unbounded region $\bot$, the number of times an idling transition is taken should reflect how 'big' the delay τ_n is.

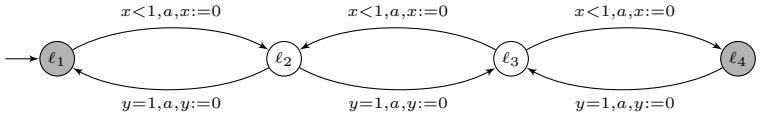
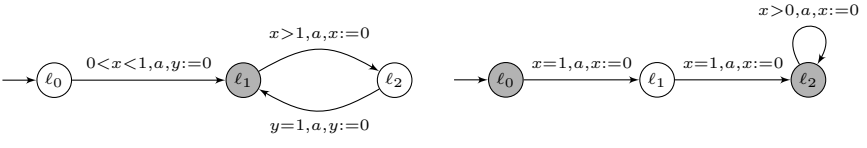


Fig. 4. A counterexample with two clocks for Lemma 6

Lemma 7 (From $\mathcal{A}^F_{cp}$ to $\mathcal{A}$, reward-diverging case). *For every reward-diverging run π in $\mathcal{A}^F_{cp}$, there exists a non-Zeno run ϱ in $\mathcal{A}$ such that $\pi \in \mathsf{Proj}_{cp}(\varrho)$ and $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$.*

Proof (Sketch). The key ingredient is that given a reward-diverging run π in $\mathcal{A}^F_{cp}$, for every $\varepsilon > 0$, one can build a non-Zeno run ϱ_ε of $\mathcal{A}$ with the following strong property: for all $n \in \mathbb{N}$, the valuation of the n-th state along ϱ_ϵ is $\frac{\epsilon}{2^n}$-close to the abstract valuation in the corresponding state in π. The accumulated reward along π diverges, hence $\mathsf{freq}_{\mathcal{A}}(\varrho_\varepsilon)$ is equal to $\mathsf{Rat}(\pi)$. □

The restriction to single-clock timed automata is crucial in Lemma 7. Indeed, consider the two-clocks timed automaton depicted in Fig. 5(a). In its corner-point abstraction there exists a reward-diverging run π with $\mathsf{Rat}(\pi) = 0$, however every run ϱ satisfies $\mathsf{freq}_{\mathcal{A}}(\varrho) > 0$.



(a) A counterexample with two clocks (b) Zeno case

Fig. 5. Counterexamples to extensions of Lemma 7

Lemma 8 (From $\mathcal{A}^F_{cp}$ to $\mathcal{A}$, reward-converging case). *For every reward-converging run π in $\mathcal{A}^F_{cp}$, if $\mathsf{Rat}(\pi) > 0$, then for every $\varepsilon > 0$, there exists a Zeno run ϱ_ε in $\mathcal{A}$ such that $\pi \in \mathsf{Proj}_{cp}(\varrho_\varepsilon)$ and $|\mathsf{freq}_{\mathcal{A}}(\varrho_\varepsilon) - \mathsf{Rat}(\pi)| < \varepsilon$.*

Proof (Sketch). A construction similar to the one used in the proof of Lemma 7 is performed. Note however that the result is slightly weaker, since in the reward-converging case, one cannot neglect imprecisions (even the smallest) forced *e.g.*, by the prohibition of the zero delays. □

Note that Lemma 8 does not hold in case $\mathsf{Rat}(\pi) = 0$, where we can only derive that the set of frequencies of runs ϱ such that $\pi \in \mathsf{Proj}_{cp}(\varrho)$ is either $\{0\}$ or $\{1\}$ or included in $(0, 1)$. Also an equivalent to Lemma 7 for Zeno runs (even in the single-clock case!) is hopeless. The timed automaton $\mathcal{A}$ depicted in Fig. 5, where $F = \{\ell_0, \ell_2\}$ is a counterexample. Indeed, in $\mathcal{A}^F_{cp}$ there is a reward-converging run π with $\mathsf{Rat}(\pi) = \frac{1}{2}$, whereas all Zeno runs in $\mathcal{A}$ have frequency larger than $\frac{1}{2}$.

3.3 Set of Frequencies of Runs in $\mathcal{A}$

We use the strong relation between frequencies in $\mathcal{A}$ and ratios in $\mathcal{A}^F_{cp}$ proven in the previous subsection to establish key properties of the set of frequencies.

Theorem 9. *Let $\mathcal{F}_{\mathcal{A}} = \{\mathsf{freq}_{\mathcal{A}}(\varrho) \mid \varrho$ run of $\mathcal{A}\}$ be the set of frequencies of runs in $\mathcal{A}$. We can compute $\inf \mathcal{F}_{\mathcal{A}}$ and $\sup \mathcal{F}_{\mathcal{A}}$. Moreover we can decide whether these bounds are reached or not. Everything can be done in NLOGSPACE.*

The above theorem is based on the two following lemmas dealing respectively with the set of non-Zeno and Zeno runs in $\mathcal{A}$.

Lemma 10 (non-Zeno case). *Let $\{C_1, \cdots, C_k\}$ be the set of reachable SCCs of $\mathcal{A}^F_{cp}$. The set of frequencies of non-Zeno runs of $\mathcal{A}$ is then $\cup_{1 \leq i \leq k}[m_i, M_i]$ where m_i (resp. M_i) is the minimal (resp. maximal) ratio for a reward-diverging cycle in C_i.*

Proof (Sketch). First, the set of ratios of reward-diverging runs in $\mathcal{A}^F_{cp}$ is exactly $\cup_{1 \leq i \leq k}[m_i, M_i]$. Indeed, given two extremal cycles c_m and c_M of ratios m and M in an SCC C of $\mathcal{A}^F_{cp}$, we show that every ratio $m \leq r \leq M$ can be obtained as the ratio of a run ending in C by combining in a proper manner c_m and c_M. Then, using Lemmas 6 and 7 we derive that the set of frequencies of non-Zeno runs in $\mathcal{A}$ coincides with the set of ratios of reward-diverging runs in $\mathcal{A}^F_{cp}$. □

Lemma 11 (Zeno case). *Given π a reward-converging run in $\mathcal{A}^F_{cp}$, it is decidable whether there exists a Zeno run ϱ such that π is the contraction of ϱ and $\mathsf{freq}_{\mathcal{A}}(\varrho) = \mathsf{Rat}(\pi)$.*

Proof (Sketch). Observe that every fragment of π between reset transitions can be considered independently, since compensations cannot occur in Zeno runs: even the smallest deviation (such as a delay ε in $\mathcal{A}$ instead of a cost 0 in π) will introduce a difference between the ratio and the frequency. A careful inspection of cases allows one to establish the result stated in the lemma. □

Using Lemmas 10 and 11, let us briefly explain how we derive Theorem 9. For each SCC C of the corner-point abstraction $\mathcal{A}^F_{cp}$, the bounds of the set of frequencies of runs whose contraction ends up in C can be computed thanks to the above lemmas. We can also furthermore decide whether these bounds can be obtained by a real run in $\mathcal{A}$. The result for the global automaton follows.

Remark 12. The link between $\mathcal{A}$ and $\mathcal{A}^F_{cp}$ differs in several aspects from [8]. First, a result similar to Lemma 6 was proven, but the runs π and π' were not in $\mathsf{Proj}_{cp}(\varrho)$, and more importantly it heavily relied on the reward-diverging hypothesis. Then the counter-part of Theorem 9 was weaker in [8] as there was no way to decide whether the bounds were reachable or not.

4 Emptiness and Universality Problems

The emptiness problem. In our context, the emptiness problem asks, given a timed automaton $\mathcal{A}$ whether there is a timed word which is accepted by $\mathcal{A}$ with

positive frequency. We also consider variants where we focus on non-Zeno or Zeno timed words. As a consequence of Theorem 9, we get the following result.

Theorem 13. *The emptiness problem for infinite (resp. non-Zeno, Zeno) timed words in single-clock timed automata is decidable. It is furthermore NLOGSPACE-Complete.*

Note that the problem is open for timed automata with 2 clocks or more.

The universality problem. We now focus on the universality problem, which asks, whether all timed words are accepted with positive frequency in a given timed automaton. We also consider variants thereof which distinguish between Zeno and non-Zeno timed words. Note that these variants are incomparable: there are timed automata that, with positive frequency, recognize all Zeno timed words but not all non-Zeno timed words, and *vice-versa*.

A first obvious result concerns deterministic timed automata. One can first check syntactically whether all infinite timed words can be read (just locally check that the automaton is complete). Then we notice that considering all timed words exactly amounts to considering all runs. Thanks to Theorem 9, one can decide, in this case, whether there is or not a run of frequency 0. If not, the automaton is universal, otherwise it is not universal.

Theorem 14. *The universality problem for infinite (resp. non-Zeno, Zeno) timed words in deterministic single-clock timed automata is decidable. It is furthermore NLOGSPACE-Complete.*

Remark 15. Note that results similar to Theorems 13 and 14 hold when considering languages defined with a threshold λ on the frequency.

If we relax the determinism assumption this becomes much harder!

Theorem 16. *The universality problem for infinite (resp. non-Zeno, Zeno) timed words in a one-clock timed automaton is non-primitive recursive. If two clocks are allowed, this problem is undecidable.*

Proof (Sketch). The proof is done by reduction to the universality problem for finite words in timed automata (which is known to be undecidable for timed automata with two clocks or more [2] and non-primitive recursive for one-clock timed automata [13]). Given a timed automaton $\mathcal{A}$ that accepts finite timed words, we construct a timed automaton $\mathcal{B}$ with an extra letter c which will be interpreted with positive frequency. From all accepting locations of $\mathcal{A}$, we allow $\mathcal{B}$ to read c and then accept everything (with positive frequency). The construction is illustrated on Fig. 6. It is easy to check that $\mathcal{A}$ is universal over Σ iff $\mathcal{B}$ is universal over $\Sigma \cup \{c\}$. □

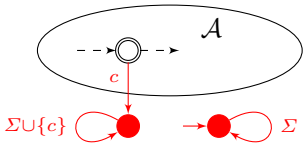


Fig. 6.

Theorem 17. *The universality problem for Zeno timed words with positive frequency in a single-clock timed automaton is decidable.*

Proof (Sketch). This decidability result is rather involved and requires some technical developments for which there is no room here. It is based on the idea that for a Zeno timed word to be accepted with positive frequency it is (necessary and) sufficient to visit an accepting location once. Furthermore the sequence of timestamps associated with a Zeno timed word is converging, and we can prove that from some point on, in the automaton, all guards will be trivially either verified or denied: for instance if the value of the clock is 1.4 after having read a prefix of the word, and if the word then converges in no more than 0.3 time units, then only the constraint $1 < x < 2$ will be satisfied while reading the suffix of the word, unless the clock is reset, in which case only the constraint $0 < x < 1$ will be satisfied. Hence the algorithm is composed of two phases: first we read the prefix of the word (and we use a now standard abstract transition system to do so, see [13]), and then for the tail of the Zeno words, the behaviour of the automaton can be reduced to that of a finite automaton (using the above argument on tails of Zeno words). □

5 Conclusion

In this paper we introduced a notion of (positive-)frequency acceptance for timed automata and studied the related emptiness and universality problems. This semantics is not comparable to the classical Büchi semantics. For deterministic single-clock timed automata, emptiness and universality are decidable by investigating the set of possible frequencies based on the corner-point abstraction. For (non-deterministic) single-clock timed automata, the universality problem restricted to Zeno timed words is decidable but non-primitive recursive. The restriction to single-clock timed automata is justified on the one hand by the undecidability of the universality problem in the general case. On the other hand, the techniques we employ to study the set of possible frequencies do not extend to timed automata with several clocks. A remaining open question is the decidability status of the universality problem for non-Zeno timed words, which is only known to be non-primitive recursive. Further investigations include a deeper study of frequencies in timed automata with multiple clocks, and also the extension of this work to languages accepted with a frequency larger than a given threshold.

References

1. Alur, R., Degorre, A., Maler, O., Weiss, G.: On omega-languages defined by mean-payoff conditions. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 333–347. Springer, Heidelberg (2009)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)

3. Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 49–62. Springer, Heidelberg (2001)
4. Asarin, E., Degorre, A.: Volume and entropy of regular timed languages: Discretization approach. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 69–83. Springer, Heidelberg (2009)
5. Baier, C., Bertrand, N., Bouyer, P., Brihaye, T., Größer, M.: Almost-sure model checking of infinite paths in one-clock timed automata. In: Proc. 23rd Annual IEEE Symp. on Logic in Computer Science (LICS 2008), pp. 217–226. IEEE Computer Society Press, Los Alamitos (2008)
6. Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J., Vaandrager, F.W.: Minimum-cost reachability for priced timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 147–161. Springer, Heidelberg (2001)
7. Bianco, A., Faella, M., Mogavero, F., Murano, A.: Quantitative fairness games. In: Proc. 8th Workshop on Quantitative Aspects of Programming Languages (QAPL 2010). ENTCS, vol. 28, pp. 48–63 (2010)
8. Bouyer, P., Brinksma, E., Larsen, K.G.: Optimal infinite scheduling for multi-priced timed automata. Formal Methods in System Design 32(1), 3–23 (2008)
9. Chatterjee, K., Doyen, L., Edelsbrunner, H., Henzinger, T.A., Rannou, P.: Mean-payoff automaton expressions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 269–283. Springer, Heidelberg (2010)
10. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. ACM Transactions on Computational Logic 11(4) (2010)
11. Kwiatkowska, M.Z., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. Theoretical Computer Science 282, 101–150 (2002)
12. Laroussinie, F., Markey, N., Schnoebelen, P.: Model checking timed automata with one or two clocks. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 387–401. Springer, Heidelberg (2004)
13. Ouaknine, J., Worrell, J.: On the decidability of Metric Temporal Logic. In: Proc. 20th Annual IEEE Symp. on Logic in Computer Science (LICS 2005), pp. 188–197. IEEE Computer Society Press, Los Alamitos (2005)
14. Tracol, M., Baier, C., Größer, M.: Recurrence and transience for probabilistic automata. In: Proc. 29th IARCS Annual Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2009). LIPIcs, vol. 4, pp. 395–406. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2009)

Büchi Automata Can Have Smaller Quotients

Lorenzo Clemente

LFCS, School of Informatics, University of Edinburgh, UK

Abstract. We study novel simulation-like preorders for quotienting nondeterministic Büchi automata. We define *fixed-word delayed simulation*, a new preorder coarser than delayed simulation. We argue that fixed-word simulation is the coarsest forward simulation-like preorder which can be used for quotienting Büchi automata, thus improving our understanding of the limits of quotienting. Also, we show that computing fixed-word simulation is PSPACE-complete.

On the practical side, we introduce *proxy simulations*, which are novel polynomial-time computable preorders sound for quotienting. In particular, *delayed proxy simulation* induce quotients that can be smaller by an arbitrarily large factor than direct backward simulation. We derive proxy simulations as the product of a theory of refinement transformers: A *refinement transformer* maps preorders nondecreasingly, preserving certain properties. We study under which general conditions refinement transformers are sound for quotienting.

1 Introduction

Büchi automata minimization is an important topic in automata theory, both for the theoretical understanding of automata over infinite words and for practical applications. Minimizing an automaton means reducing the number of its states as much as possible, while preserving the recognized language. Minimal automata need not be unique, and their structure does not necessarily bear any resemblance to the original model; in the realm of infinite words, this holds even for deterministic models. This hints at why exact minimization has high complexity: Indeed, minimality checking is PSPACE-hard for nondeterministic models (already over finite words [13]), and NP-hard for deterministic Büchi automata [20]. Moreover, even approximating the minimal model is hard [9].

By posing suitable restrictions on the minimization procedure, it is nonetheless possible to trade exact minimality for efficiency. In the approach of *quotienting*, smaller automata are obtained by merging together equivalent states, under appropriately defined equivalences. In particular, quotienting by simulation equivalence has proven to be an effective heuristics for reducing the size of automata in cases of practical relevance.

The notion of *simulation preorder* and *equivalence* [18] is a crucial tool for comparing the behaviour of systems. It is best described via a game between two players, Duplicator and Spoiler, where the former tries to stepwise match the moves of the latter. But not every simulation preorder can be used for quotienting: We call a preorder *good for quotienting* (GFQ) if the quotient automaton (w.r.t. the induced equivalence) recognizes the same language as the original automaton. In particular, a necessary condition for a simulation to be GFQ is to take into account the acceptance condition: For

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 258–270, 2011.

example, in *direct* simulation [5], Duplicator has the additional requirement to visit an accepting state whenever Spoiler does so, while in the coarser *fair* simulation [11], Duplicator has to visit infinitely many accepting states if Spoiler does so. But, while direct simulation is GFQ [2], fair simulation is not [12].[1] This prompted the development of *delayed* simulation [7], a GFQ preorder intermediate between direct and fair simulation.

We study the border of GFQ preorders. In our first attempt we generalize delayed simulation to *delayed containment*. While in simulation the two players take turns in selecting transitions, in containment the game ends in one round: First Spoiler picks an infinite path, and then Duplicator has to match it with another infinite path. The winning condition is delayed-like: Every accepting state of Spoiler has to be matched by an accepting state of Duplicator, possibly occurring later. Therefore, in delayed containment Duplicator is much stronger than in simulation; in other words, containment is coarser than simulation. In fact, it is *too coarse*: We give a counterexample where delayed containment is not GFQ. We henceforth turn our attention to finer preorders.

In our second attempt, we remedy to the deficiency above by introducing *fixed-word delayed simulation*, an intermediate notion between simulation and containment. In fixed-word simulation, Spoiler does not reveal the whole path in advance like in containment; instead, she only declares the input word beforehand. Then, the simulation game starts, but now transitions can be taken only if they match the word fixed earlier by Spoiler. Unlike containment, fixed-word delayed simulation is GFQ, as we show.

We proceed by looking at even coarser GFQ preorders. We enrich fixed-word simulation by allowing Duplicator to use *multiple pebbles*, in the style of [6]. The question arises as whether Duplicator gains more power by "hedging her bets" when she already knows the input word in advance. By using an ordinal ranking argument (reminiscent of [16]), we establish that this is not the case, and that the multipebble hierarchy collapses to the 1-pebble case, i.e., to fixed-word delayed simulation itself. Incidentally, this also shows that the whole delayed multipebble hierarchy from [6] is entirely contained in fixed-word delayed simulation—the containment being strict.

For what concerns the complexity of computing fixed-word simulation, we establish that it is PSPACE-complete, by a mutual reduction from Büchi automata universality.

With the aim of getting tractable preorders, we then look at a different way of obtaining GFQ relations, by introducing a theory of refinement transformers: A *refinement transformer* maps a preorder $\preceq$ to a coarser preorder $\preceq'$, s.t., once $\preceq$ is known, $\preceq'$ can be computed with only a polynomial time overhead. The idea is to play a simulation-like game, where we allow Duplicator to "jump" to $\preceq$-bigger states, called *proxies*, after Spoiler has selected her transition. Duplicator can then reply with a transition from the proxy instead of the original state. We say that proxy states are *dynamic* in the sense that they depend on the transition selected by Spoiler.[2] Under certain conditions, we show that refinement transformers induce GFQ preorders.

Finally, we introduce *proxy simulations*, which are novel polynomial time GFQ preorders obtained by applying refinement transformers to a concrete preorder $\preceq$, namely, to *backward* direct simulation (called reverse simulation in [21]). We define two versions of proxy simulation, direct and delayed, the latter being coarser than the former,

[1] In fact, for Büchi automata it is well-known that also language equivalence is not GFQ.

[2] Proxies are strongly related to mediators [1]. We compare them in depth in Section 6.

and both coarser than direct backward simulation. Moreover, we show that the delayed variant can achieve quotients smaller than direct proxy simulation by an arbitrarily large factor. Full proofs can be found in the technical report [3].

Related work. Delayed simulation [7] has been extended to generalized automata [14], to multiple pebbles [6], to alternating automata [8] and to the combination of the last two [4]. Fair simulation has been used for state space reduction in [10]. The abstract idea of mixing forward and backward modes in quotienting can be traced back at least to [19]; in the context of alternating automata, it has been studied in [1].

2 Preliminaries

Games. For a finite sequence $\pi = e_0 e_1 \cdots e_{k-1}$, let $|\pi| = k$ be its length, and let $\mathrm{last}(\pi) = e_{k-1}$ be its last element. If π is infinite, then take $|\pi| = \omega$.

A *game* is a tuple $G = (P, P_0, P_1, p_I, \Gamma, \Gamma_0, \Gamma_1, W)$, where P is the set of positions, partitioned into disjoint sets P_0 and P_1, $p_I \in P_0$ is the initial position, $\Gamma = \Gamma_0 \cup \Gamma_1$ is the set of moves, where $\Gamma_0 \subseteq P_0 \times P_1$ and $\Gamma_1 \subseteq P_1 \times P_0$ are the set of moves of Player 0 and Player 1, respectively, and $W \subseteq P_0^\omega$ is the winning condition. A *path* is a finite or infinite sequence of states $\pi = p_0^0 p_0^1 p_1^0 p_1^1 \cdots$ starting in p_I, such that, for all $i < |\pi|$, $(p_i^0, p_i^1) \in \Gamma_0$ and $(p_i^1, p_{i+1}^0) \in \Gamma_1$. *Partial plays* and *plays* are finite and infinite paths, respectively. We assume that there are no dead ends in the game. A play is *winning* for Player 1 iff $p_0^0 p_1^0 p_2^0 \cdots \in W$; otherwise, is it winning for Player 0.

A *strategy* for Player 0 is a partial function $\sigma_0 : (P_0 P_1)^* P_0 \mapsto P_1$ s.t., for any partial play $\pi \in (P_0 P_1)^* P_0$, if σ_0 is defined on π, then $\pi \cdot \sigma_0(\pi)$ is again a partial play. A play π is *σ_0-conform* iff, for every $i \geq 0$, $p_i^1 = \sigma_0(p_0^0 p_0^1 \cdots p_i^0)$. Similarly, a strategy for Player 1 is a partial function $\sigma_1 : (P_0 P_1)^+ \mapsto P_0$ s.t., for any partial play $\pi \in (P_0 P_1)^+$, if σ_1 is defined on π, then $\pi \cdot \sigma_1(\pi)$ is again a partial play. A play π is σ_1-conform iff, for every $i \geq 0$, $p_{i+1}^0 = \sigma_0(p_0^0 p_0^1 \cdots p_i^0 p_i^1)$. While we do not require strategies to be total functions, we do require that a strategy σ is defined on all σ-conform partial plays.

A strategy σ_i is a *winning strategy* for Player i iff all σ_i-conform plays are winning for Player i. We say that Player i wins the game G if she has a winning strategy.

Automata. A *nondeterministic Büchi automaton* (NBA) is a tuple $\mathcal{Q} = (Q, \Sigma, I, \Delta, F)$, where Q is a finite set of states, Σ is a finite alphabet, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states and $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation. We also write $q \xrightarrow{a} q'$ instead of $(q, a, q') \in \Delta$, and just $q \longrightarrow q'$ when $\exists a \in \Sigma \cdot q \xrightarrow{a} q'$. For two sets of states $\mathbf{q}, \mathbf{q}' \subseteq Q$, we write $\mathbf{q} \overset{a}{\Longrightarrow} \mathbf{q}'$ iff $\forall q' \in \mathbf{q}' \cdot \exists q \in \mathbf{q} \cdot q \xrightarrow{a} q'$.[3] For a state $q \in Q$, let $[q \in F] = 1$ if q is accepting, and 0 otherwise. We assume that every state is reachable from some initial state, and that the transition relation is total.

For a finite or infinite sequence of states $\rho = q_0 q_1 \cdots$ and an index $i \leq |\rho|$, let $\mathsf{cnt\text{-}final}(\rho, i)$ be the number of final states occuring in ρ up to (and including) the i-th element. Formally, $\mathsf{cnt\text{-}final}(\rho, i) = \sum_{0 \leq k < i} [q_k \in F]$, with $\mathsf{cnt\text{-}final}(\rho, 0) = 0$. Let $\mathsf{cnt\text{-}final}(\rho) = \mathsf{cnt\text{-}final}(\rho, |\rho|)$. If ρ is infinite, then $\mathsf{cnt\text{-}final}(\rho) = \omega$ iff ρ contains infinitely many accepting states.

[3] This kind of backward-compatible transition had already appeared in [17].

Fix a finite or infinite word $w = a_0 a_1 \cdots$. A *path* π *over* w is a sequence $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \cdots$ of length $|w|+1$. A path is *initial* if it starts in an initial state $q_0 \in I$, it is a *run* if it is initial and infinite, and it is *fair* if $\mathsf{cnt\text{-}final}(\pi) = \omega$. An *accepting run* is a run which is fair. The *language* $\mathcal{L}^\omega(\mathcal{Q})$ of a NBA $\mathcal{Q}$ is the set of infinite words which admit an accepting run, i.e., $\mathcal{L}^\omega(\mathcal{Q}) = \{w \in \Sigma^\omega \mid \text{ there exists an accepting run } \pi \text{ over } w\}$.

Quotients. Let $\mathcal{Q} = (Q, \Sigma, I, \Delta, F)$ be a NBA and let R be any binary relation on Q. We say that $\approx_R$ is the *equivalence induced by* R if $\approx_R$ is the largest equivalence contained in the transitive and reflexive closure of R. I.e., $\approx_R = R^* \cap (R^*)^{-1}$. Let the function $[\cdot]_R : Q \mapsto 2^Q$ map each element $q \in Q$ to the equivalence class $[q]_R \subseteq Q$ it belongs to, i.e., $[q]_R := \{q' \in Q \mid q \approx_R q'\}$. We overload $[P]_R$ on sets $P \subseteq Q$ by taking the set of equivalence classes. When clear from the context, we avoid noting the dependence of $\approx$ and $[\cdot]$ on R.

An equivalence $\approx$ on $\mathcal{Q}$ induces the *quotient automaton* $\mathcal{Q}_\approx = ([Q], \Sigma, [I], \Delta_\approx, [F])$, where, for any $q, q' \in Q$ and $a \in \Sigma$, $([q], a, [q']) \in \Delta_\approx$ iff $(q, a, q') \in \Delta$. This is called a *naïve* quotient since both initial/final states and transitions are induced representative-wise. When we quotient w.r.t. a relation R which is not itself an equivalence, we actually mean quotenting w.r.t. the induced equivalence $\approx$. We say that R is *good for quotienting* (GFQ) if quotienting $\mathcal{Q}$ w.r.t. R preserves the language, that is, $\mathcal{L}^\omega(\mathcal{Q}) = \mathcal{L}^\omega(\mathcal{Q}_\approx)$.

Lemma 1. *For two equivalences* $\approx_0, \approx_1$*, if* $\approx_0 \subseteq \approx_1$*, then* $\mathcal{L}^\omega(\mathcal{Q}_{\approx_0}) \subseteq \mathcal{L}^\omega(\mathcal{Q}_{\approx_1})$*. In particular, by letting* $\approx_0$ *be the identity,* $\mathcal{L}^\omega(\mathcal{Q}) \subseteq \mathcal{L}^\omega(\mathcal{Q}_{\approx_1})$*.*

3 Quotienting with Forward Simulations

In this section we study several generalizations of delayed simulation, in order to investigate the border of good for quotienting (GFQ) forward-like preorders. In our first attempt we introduce *delayed containment*, which is obtained as a modification of the usual simulation interaction between players: In the delayed containment game between q and s there are only two rounds. Spoiler moves first and selects both an infinite word $w = a_0 a_1 \cdots$ and an infinite path $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots$ over w starting in $q = q_0$; then, Duplicator replies with an infinite path $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ over w starting in $s = s_0$. The winning condition is delayed-like: $\forall i \cdot q_i \in F \implies \exists j \geq i \cdot s_j \in F$. If Duplicator wins the delayed containment game between q and s, we write $q \subseteq^{\text{de}} s$. Clearly, $\subseteq^{\text{de}}$ is a preorder implying language containment. One might wonder whether delayed-containment is GFQ. Unfortunately, this is not the case (see Figure 5 in the tech. rep. [3]). Therefore, $\subseteq^{\text{de}}$ is too coarse for quotienting, and we shall look at finer relations.

Lemma 2. $\subseteq^{de}$ *is not a GFQ preorder.*

3.1 Fixed-Word Delayed Simulation

Our second attempt at generalizing delayed simulation still retains the flavour of containment. While in containment $\subseteq^{\text{de}}$ Spoiler reveals both the input word w and a path over w, in *fixed-word* simulation $\sqsubseteq^{\text{de}}_{\text{fx}}$ Spoiler reveals w only. Then, after w has been fixed, the game proceeds like in delayed simulation, with the proviso that transitions match symbols in w.[4] Formally, let $w = a_0 a_1 \cdots \in \Sigma^\omega$. In the w-simulation game

[4] The related notion of fixed-word *fair* simulation clearly coincides with ω-language inclusion.

$G_w^{\text{de}}(q,s)$ the set of positions of Spoiler is $P_0 = Q \times Q \times \mathbb{N}$, the set of positions of Duplicator is $P_1 = Q \times Q \times Q \times \mathbb{N}$ and $\langle q, s, 0\rangle$ is the initial position. Transitions are determined as follows: Spoiler can select a move of the form $(\langle q, s, i\rangle, \langle q, s, q', i\rangle) \in \Gamma_0^{w\text{-de}}$ if $q \xrightarrow{a_i} q'$, and Duplicator can select a move of the form $(\langle q, s, q', i\rangle, \langle q', s', i+1\rangle) \in \Gamma_1^{w\text{-de}}$ if $s \xrightarrow{a_i} s'$. Notice that the input symbol a_i is fixed, and it has to match the corresponding symbol in w. The winning condition is $W = \{\langle q_0, s_0, 0\rangle\langle q_1, s_1, 1\rangle \cdots \mid \forall i \cdot q_i \in F \implies \exists j \geq i \cdot s_j \in F\}$. Let $q \sqsubseteq_w^{\text{de}} s$ iff Duplicator wins the w-simulation game $G_w^{\text{de}}(q,s)$, and $q \sqsubseteq_{\text{fx}}^{\text{de}} s$ iff $q \sqsubseteq_w^{\text{de}} s$ for all $w \in \Sigma^\omega$. Clearly, fixed-word simulation is a preorder implying containment.

Fact 1. *$\sqsubseteq_{fx}^{de}$ is a reflexive and transitive relation, and $\forall q, s \in Q \cdot q \sqsubseteq_{fx}^{de} s \implies q \subseteq^{de} s$.*

Unlike delayed containment, fixed-word delayed simulation is GFQ. Moreover, fixed-word delayed simulation quotients can be more succint than (multipebble) delayed simulation quotients by an arbitrarily large factor. See Figure 6 in [3].

Theorem 1. *$\sqsubseteq_{fx}^{de}$ is good for quotienting.*

Complexity of delayed fixed word simulation. Let q, s be two states in $\mathcal{Q}$. We reduce the problem of checking $q \sqsubseteq_{\text{fx}}^{\text{de}} s$ to the universality problem of a suitable alternating Büchi product automaton (ABA) $\mathcal{A}$. We design $\mathcal{A}$ to accept exactly those words w s.t. Duplicator wins $G_w^{\text{de}}(q,s)$. Then, by the definition of $\sqsubseteq_{\text{fx}}^{\text{de}}$, it is enough to check whether $\mathcal{A}$ has universal language. See [22] (or Appendix A.1 [3]) for background on ABAs.

The idea is to enrich configurations in the fixed-word simulation game by adding an obligation bit recording whether Duplicator has any pending constraint to visit an accepting state. Initially the bit is 0, and it is set to 1 whenever Spoiler is accepting; a reset to 0 can occur afterwards, if and when Duplicator visits an accepting state.

Let $\mathcal{Q} = (Q, \Sigma, I, \Delta, F)$ be a NBA. We define a product ABA $\mathcal{A} = (A, \Sigma, \delta, \alpha)$ as follows: The set of states is $A = Q \times Q \times \{0,1\}$, final states are of the form $\alpha = Q \times Q \times \{0\}$ and, for any $\langle q, s, b\rangle \in A$ and $a \in \Sigma$,

$$\delta(\langle q, s, b\rangle, a) = \bigwedge_{q \xrightarrow{a} q'} \bigvee_{s \xrightarrow{a} s'} \langle q', s', b'\rangle, \qquad \text{where } b' = \begin{cases} 0 & \text{if } s \in F \\ 1 & \text{if } q \in F \wedge s \notin F \\ b & \text{otherwise} \end{cases}$$

It follows directly from the definitions that $q \sqsubseteq_{\text{fx}}^{\text{de}} s$ iff $\mathcal{L}^\omega(\langle q, s, 0\rangle) = \Sigma^\omega$. A reduction in the other direction is immediate already for NBAs: In fact, an NBA $\mathcal{Q}$ is universal iff $\mathcal{U} \sqsubseteq_{\text{fx}}^{\text{de}} \mathcal{Q}$, where $\mathcal{U}$ is the trivial, universal one-state automaton with an accepting Σ-loop. It is well-known that universality is PSPACE-complete for ABAs/NBAs [15].

Theorem 2. *Computing fixed-word delayed simulation is PSPACE-complete.*

3.2 Multipebble Fixed-Word Delayed Simulation

Having established that fixed-word simulation is GFQ, the next question is whether we can find other natural GFQ preorders between fixed-word and delayed containment. A natural attempt is to add a multipebble facility on top of $\sqsubseteq_{\text{fx}}^{\text{de}}$. Intuitively, when Duplicator uses multiple pebbles she can "hedge her bets" by moving pebbles to several successors. This allows Duplicator to delay committing to any particular choice by arbitrarily many steps: In particular, she can always gain knowledge on any *finite* number

of moves by Spoiler. Perhaps surprisingly, we show that *Duplicator does not gain more power by using pebbles*. This is stated in Theorem 3, and it is the major technical result of this section. It follows that, once Duplicator knows the input word in advance, there is no difference between knowing only the next step by Spoiler, or the next l steps, for any finite $l > 1$. Yet, if we allow $l = \omega$ lookahead, then we recover delayed containment $\subseteq^{\text{de}}$, which is not GFQ by Lemma 2. Therefore, w.r.t. to the degree of lookahead, $\sqsubseteq^{\text{de}}_{\text{fx}}$ is the coarsest GFQ relation included in $\subseteq^{\text{de}}$.

We now define the multipebble fixed-word delayed simulation. Let $k \geq 1$ and $w = a_0 a_1 \cdots \in \Sigma^\omega$. In the k-multipebble w-delayed simulation game $G_w^{k\text{-de}}(q, s)$ the set of positions of Spoiler is $Q \times 2^Q \times \mathbb{N}$, the set of positions of Duplicator is $Q \times 2^Q \times Q \times \mathbb{N}$, the initial position is $\langle q, \{s\}, 0\rangle$, and transitions are: $(\langle q, \mathbf{s}, i\rangle, \langle q, \mathbf{s}, q', i\rangle) \in \Gamma_0$ iff $q \xrightarrow{a_i} q'$, and $(\langle q, \mathbf{s}, q', i\rangle, \langle q', \mathbf{s}', i+1\rangle) \in \Gamma_1$ iff $\mathbf{s} \overset{a_i}{\Longrightarrow} \mathbf{s}'$ and $|\mathbf{s}'| \leq k$.

Before defining the winning set we need some preparation. Given an infinite sequence $\pi = \langle q_0, \mathbf{s}_0, 0\rangle\langle q_1, \mathbf{s}_1, 1\rangle \cdots$ over $w = a_0 a_1 \cdots$ and a round $j \geq 0$, we say that a state $s \in \mathbf{s}_j$ *has been accepting* since some previous round $i \leq j$, written $\mathsf{accepting}_j^i(s, \pi)$, iff either $s \in F$, or $i < j$ and there exists $\hat{s} \in \mathbf{s}_{j-1}$ s.t. $\hat{s} \xrightarrow{a_{j-1}} s$ and $\mathsf{accepting}_{j-1}^i(\hat{s}, \pi)$. We say that $\mathbf{s}_j$ is *good since round* $i \leq j$, written $\mathsf{good}_j^i(\mathbf{s}_j, \pi)$, iff at round j every state $s \in \mathbf{s}_j$ has been accepting since round i, and j is the least round for which this holds [6]. Duplicator wins a play if, whenever $q_i \in F$ there exists $j \geq i$ s.t. $\mathsf{good}_j^i(\mathbf{s}_j, \pi)$. We write $q \sqsubseteq_w^{k\text{-de}} s$ iff Duplicator wins $G_w^{k\text{-de}}(q, s)$, and we write $q \sqsubseteq_{\text{fx}}^{k\text{-de}} s$ iff $\forall w \in \Sigma^\omega \cdot q \sqsubseteq_w^{k\text{-de}} s$.

Clearly, pebble simulations induce a non-decreasing hierarchy: $\sqsubseteq_{\text{fx}}^{1\text{-de}} \subseteq \sqsubseteq_{\text{fx}}^{2\text{-de}} \subseteq \cdots$. We establish that the hierarchy actually collapses to the $k = 1$ level. This result is non-trivial, since the delayed winning condition requires reasoning not only about the *possibility* of Duplicator to visit accepting states in the future, but also about exactly *when* such a visit occurs. Technically, our argument uses a ranking argument similar to [16] (see Appendix A.2 [3]), with the notable difference that our ranks are *ordinals* ($\leq \omega^2$), instead of natural numbers. We need ordinals to represent how long a player can delay visiting accepting states, and how this events nest with each other. Finally, notice that the result above implies that the multipebble delayed simulation hierarchy of [6] is entirely contained in $\sqsubseteq_{\text{fx}}^{\text{de}}$, and the containment is strict (Fig. 6 [3]).

Theorem 3. *For any NBA $\mathcal{Q}$, $k \geq 1$ and states $q, s \in Q$, $q \sqsubseteq_{fx}^{k\text{-}de} s$ iff $q \sqsubseteq_{fx}^{de} s$.*

4 Jumping-Safe Relations

In this section we present the general technique which is used throughout the paper to establish that preorders are GFQ. We introduce *jumping-safe relations*, which are shown to be GFQ (Theorem 4). In Section 5 we use jumping-safety as an invariant when applying refinement transformers. We start off with an analysis of acceping runs.

Coherent sequences of paths. Fix an infinite word $w \in \Sigma^\omega$. Let $\Pi := \pi_0, \pi_1, \ldots$ be an infinite sequence of longer and longer *finite* initial paths in $\mathcal{Q}$ over (prefixes of) w. We are interested in finding a sufficient condition for the existence of an accepting run over w. A necessary condition is that the number of final states in π_i grows unboundedly as

i goes to ω. In the case of deterministic automata this condition is also sufficient: Indeed, in a deterministic automaton there exists a unique run over w, which is accepting exactly when the number of accepting stated visited by its prefixes goes to infinity. In this case, we say that the π_i's are *strongly coherent* since they next path extends the previous one. Unfortunately, in the general case of nondeterministic automata it is quite possible to have paths that visit arbitrarily many final states but no accepting run exists. This occurs because final states can appear arbitrarily late. Indeed, consider Figure 1. Take $w = aba^2ba^3b\cdots$: For every prefx $w_i = aba^2b\cdots a^i$ there exists a path $\pi_i = qq\cdots q\cdot s^i$ over w_i visiting a final state i times. Still, $w \notin \mathcal{L}^\omega(\mathcal{Q})$.

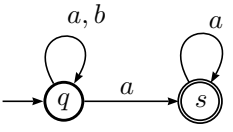


Fig. 1. Automaton $\mathcal{Q}$

Therefore, we forbid accepting states to "clump away" in the tail of the path. We ensure this by imposing the existence of an infinite sequence of indices $j_0, j_1, \cdots$ s.t., for all i, and for all k_i big enough, the number of final states in π_{k_i} up to the j_i-th state is at least i. In this way, we are guaranteed that at least i final states are present within j_i steps in all but finitely many paths.

Definition 1. *Let $\Pi := \pi_0, \pi_1, \ldots$ be an infinite sequence of finite paths. We say that Π is a* coherent sequence of paths *if the following property holds:*

$$\forall i \cdot \exists j \cdot \exists h \cdot \forall k \geq h \cdot j < |\pi_k| \wedge \mathsf{cnt\text{-}final}(\pi_k, j) \geq i \,. \tag{1}$$

Lemma 3. *If Π is coherent, then any infinite subsequence Π' thereof is coherent.*

We sketch below the proof that coherent sequences induce fair paths. Let $\Pi = \pi_0, \pi_1, \ldots$ be a coherent sequence of paths in $\mathcal{Q}$. Let $i = 1$, and let j_1 be the index witnessing Π is coherent. Since the π_k's are branches in a finitely branching tree, there are only a finite number of different prefixes of length j_1. Therefore, there exists a prefix ρ_1 which is common to infinitely many paths. Let $\Pi' = \pi'_0, \pi'_1, \ldots$ be the infinite subsequence of Π containing only suffixes of ρ_1. Clearly ρ_1 contains at least 1 final state, and each π' in Π' extends ρ_1. By Lemma 3, Π' is coherent. For $i = 2$, we can apply the reasoning again to Π', and we obtain a longer prefix ρ_2 extending ρ_1, and containing at least 2 final states. Let Π'' be the coherent subsequence of Π' containing only suffixes of ρ_2. In this fashion, we obtain an infinite sequence of *strongly* coherent (finite) paths $\rho_1, \rho_2, \cdots$ s.t. ρ_i extends ρ_{i-1} and contains at least i final states. The infinite path to which the sequence converges is the fair path we are after.

Lemma 4. *Let $w \in \Sigma^\omega$ and $\pi_0, \pi_1, \ldots$ as above. If $\pi_0, \pi_1, \ldots$ is coherent, then there exists a fair path ρ over w. Moreover, if all π_i's are initial, then ρ is initial.*

Jumping-safe relations. We established that coherent sequences induce accepting paths. Next, we introduce *jumping-safe* relations, which are designed to induce coherent sequences (and thus accepting paths) when used in quotienting. The idea is to view a path in the quotient automaton as a jumping path in the original automaton, where a "jumping path" is one that can take arbitrary jumps to equivalent states. Jumping-safe relations allows us to transform the sequence of prefixes of an accepting jumping path into a coherent sequence of non-jumping paths; by Lemma 4, this induces a (nonjumping) accepting path.

Fix a word $w = a_0 a_1 \cdots \in \Sigma^\omega$, and let R be a binary relation over Q. An *R-jumping path* is an infinite sequence

$$\pi = q_0 \; R \; q_0^F \; R \; \hat{q}_0 \xrightarrow{a_0} q_1 \; R \; q_1^F \; R \; \hat{q}_1 \xrightarrow{a_1} q_2 \cdots , \tag{2}$$

and we say that π is *initial* if $q_0 \in I$, and *fair* if $q_i^F \in F$ for infinitely many i's.

Definition 2. *A binary relation R is* jumping-safe *iff for any initial R-jumping path π there exists an infinite sequence of initial finite paths $\pi_0, \pi_1, \ldots$ over suitable prefixes of w s.t. $\mathrm{last}(\pi_i)\; R\; q_i$ and, if π is fair, then $\pi_0, \pi_1, \ldots$ is coherent.*

Theorem 4. *Jumping-safe preorders are good for quotienting.*

In Section 5 we introduce refinement transformers, which are designed to preserve jumping-safety. Then, in Section 6 we specialize the approach to *backward direct simulation* $\sqsubseteq^{\mathrm{di}}_{\mathrm{bw}}$ [21], which provides an initial jumping-safe preorder, and which we introduce next: $\sqsubseteq^{\mathrm{di}}_{\mathrm{bw}}$ is the coarsest preorder s.t. $q \sqsubseteq^{\mathrm{di}}_{\mathrm{bw}} s$ implies 1) $\forall (q' \xrightarrow{a} q) \cdot \exists (s' \xrightarrow{a} s) \cdot q' \sqsubseteq^{\mathrm{di}}_{\mathrm{bw}} s'$, 2) $q \in F \implies s \in F$, and 3) $q \in I \implies s \in I$.

Fact 2. *$\sqsubseteq^{di}_{bw}$ is jumping-safe and computable in polynomial time.*

5 Refinement Transformers

We study how to obtain GFQ preorders coarser than forward/backward simulation. As a preliminary example, notice that it is not possible to generalize simultaneously both forward and backward simulations. See the counterexample in Fig. 2, where any relation coarser than both forward and backward simulation is not GFQ. Let $\approx^{\mathrm{di}}_{\mathrm{bw}}$ and $\approx^{\mathrm{di}}_{\mathrm{fw}}$ be backward and forward direct simulation equivalence, respectively. We have $q_1 \approx^{\mathrm{di}}_{\mathrm{bw}} q_2 \approx^{\mathrm{di}}_{\mathrm{fw}} q_3$, but "glueing together" q_1, q_2, q_3 would introduce the extraneous word ba^ω. Therefore, one needs to choose whether to extend either forward or backward simulation. The former approach has been pursued in the *mediated preorders* of [1] (in the more general context of alternating automata). Here, we extend backward refinements.

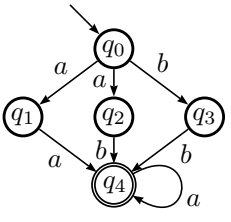


Fig. 2.

We define a *refinement transformer* τ_0 mapping a relation R to a new, coarser relation $\tau_0(R)$. We present τ_0 via a forward direct simulation-like game where Duplicator is allowed to "jump" to R-bigger states—called *proxies*. Formally, in the $\tau_0(R)$ simulation game Spoiler's positions are in $Q \times Q$, Duplicator's position are in $Q \times Q \times \Sigma \times Q$ and transitions are as follows: Spoiler picks a transition $(\langle s, q\rangle, \langle s, q, a, q'\rangle) \in \Gamma_0$ simply when $q \xrightarrow{a} q'$, and Duplicator picks a transition $(\langle s, q, a, q'\rangle, \langle s', q'\rangle) \in \Gamma_1$ iff there exists a proxy $\hat{s}$ s.t. $s \; R \; \hat{s}$ and $\hat{s} \xrightarrow{a} s'$. The winning condition is: $\forall i \geq 0 \cdot q_i \in F \implies \hat{s}_i \in F$. If Duplicator wins starting from the initial position $\langle s, q\rangle$, we write $s \; \tau_0(R) \; q$. (Notice that we swapped the usual order between q and s here.)

Lemma 5. *For a preorder R, $R \subseteq R \circ \tau_0(R) \subseteq \tau_0(R)$.*

Unfortunately, $\tau_0(R)$ is not necessarily a transitive relation. Therefore, it is not immediately clear how to define a suitable equivalence for quotienting. Figure 2 shows that taking the transitive closure of $\tau_0(R)$ is incorrect—already when R is direct backward simulation $\sqsubseteq^{\text{di}}_{\text{bw}}$: Let $\preceq = \tau_0(\sqsubseteq^{\text{di}}_{\text{bw}})$ and let $\approx = \preceq \cap \preceq^{-1}$. We have $q_3 \approx q_2 \approx q_1 \ \preceq \ q_3$, but $q_3 \not\preceq q_1$, and forcing $q_1 \approx q_3$ is incorrect, as noted earlier.

Thus, $\tau_0(R)$ is not GFQ and we need to look at its transitive fragments. Let $T \subseteq \tau_0(R)$. We say that R is *F-respecting* if $q \ R \ s \wedge q \in F \implies s \in F$, that T is *self-respecting* if Duplicator wins by never leaving T, that T is *appealing* if transitive and self-respecting, and that T *improves on* R if $R \subseteq T$.

Theorem 5. *Let R a F-respecting preorder, and let $T \subseteq \tau_0(R)$ be an appealing, improving fragment of $\tau_0(R)$. If R is jumping-safe, then T is jumping-safe.*

In particular, by Theorem 4, T is GFQ. Notice that requiring that R is GFQ is not sufficient here, and we need the stronger invariant given by jumping-safety.

Given an appealing fragment $T \subseteq \tau_0(R)$, a natural question is whether $\tau_0(T)$ improves on $\tau_0(R)$, so that τ_0 can be applied repeatedly to get bigger and bigger preorders. We see in the next lemma that this is not the case.

Lemma 6. *For any reflexive R, let $T \subseteq \tau_0(R)$ be any appealing fragment of $\tau_0(R)$. Then, $\tau_0(T) \subseteq \tau_0(R)$.*

Efficient appealing fragments. By Theorems 4 and 5, appealing fragments of τ_0 are GFQ. Yet, we have not specified any method for obtaining these. Ideally, one looks for fragments having maximal cardinality (which yelds maximal reduction under quotienting), but finding them is computationally expensive. Instead, we define a new transformer τ_1 which is guaranteed to produce only appealing fragments,[5] which, while not maximal in general, are maximal amongst all *improving* fragments (Lemma 7).

The reason why $\tau_0(R)$ is not transitive is that only Duplicator is allowed to make "R-jumps". This asymmetry is an obstacle to compose simulation games. We recover transitivity by allowing Spoiler to jump as well, thus restoring the symmetry. Formally, the $\tau_1(R)$ simulation game is identical to the one for $\tau_0(R)$, the only difference being that also Spoiler is now allowed to "jump", i.e., she can pick a transition $(\langle s, q\rangle, \langle s, q, a, q'\rangle) \in \Gamma_0$ iff there exists $\hat{q}$ s.t. $q \ R \ \hat{q}$ and $\hat{q} \xrightarrow{a} q'$. The winning condition is: $\forall i \geq 0 \cdot \hat{q}_i \in F \implies \hat{s}_i \in F$. Let $s \ \tau_1(R) \ q$ if Duplicator wins from position $\langle s, q\rangle$. It is immediate to see that $\tau_1(R)$ is an appealing fragment of $\tau_0(R)$, and that τ_1 is improving on transitive relations R's. Thus, for a preorder R, $R \subseteq \tau_1(R) \subseteq \tau_0(R)$. By Theorems 4 and 5, $\tau_1(R)$ is GFQ (if R is F-respecting).

It turns out that $\tau_1(R)$ is actually the *maximal* appealing, improving fragment of $\tau_0(R)$. This is non-obvious, since the class of appealing T's is not closed under union—still, it admits a maximal element. Therefore, τ_1 is an optimal solution to the problem of finding appealing, improving fragments of $\tau_0(R)$.

[5] τ_1 needs not be the only solution to this problem: Other ways of obtaining appealing fragments of τ_0 might exist. For this reason, we have given a separate treatment of τ_0 in its generality, together with the general correctness statement (Theorem 5).

Lemma 7. *For any R, let $T \subseteq \tau_0(R)$ be any appealing fragment of $\tau_0(R)$. If $R \subseteq T$ (i.e., R is improving), then $T \subseteq \tau_1(R)$.*

5.1 Delayed-Like Refinement Transformers

We show that the refinement transformer approach can yield relations even coarser than τ_1. Our first attempt is to generalize the direct-like winning condition of τ_0 to a delayed one. Let τ_0^{de} be the same as τ_0 except for the different winning condition, which now is: $\forall i \geq 0 \cdot q_i \in F \implies \exists j \geq i \cdot \hat{s}_j \in F$. Clearly, τ_0^{de} inherits the same transitivity issues of τ_0. Unfortunately, the approach of taking appealing fragments is not sound here, due to the weaker winning condition. See Figure 7 in [3] for a counterexample.

We overcome these issues by dropping τ_0^{de} altogether, and directly generalize τ_1 (instead of τ_0) to a delayed-like notion. The *delayed refinement transformer* τ_1^{de} is like τ_1, except for the new winning condition: $\forall i \geq 0 \cdot \hat{q}_i \in F \implies \exists j \geq i \cdot \hat{s}_j \in F$. Notice that $\tau_1^{\text{de}}(R)$ is at least as coarse as $\tau_1(R)$, and incomparable with $\tau_0(R)$. Once R is given, $\tau_1^{\text{de}}(R)$ can be computed in polynomial time. See Appendix D in the tech. rep. [3].

Lemma 8. *For any R, $\tau_1^{de}(R)$ is transitive.*

Theorem 6. *If R is a jumping-safe F-respecting preorder, then $\tau_1^{de}(R)$ is jumping-safe.*

6 Proxy Simulations

We apply the theory of transformers from Section 5 to a specific F-respecting preorder, namely backward direct simulation, obtaining *proxy simulations*. Notice that proxy simulation-equivalent states need not have the same language; yet, proxy simulations are GFQ (and computable in polynomial time).

6.1 Direct Proxy Simulation

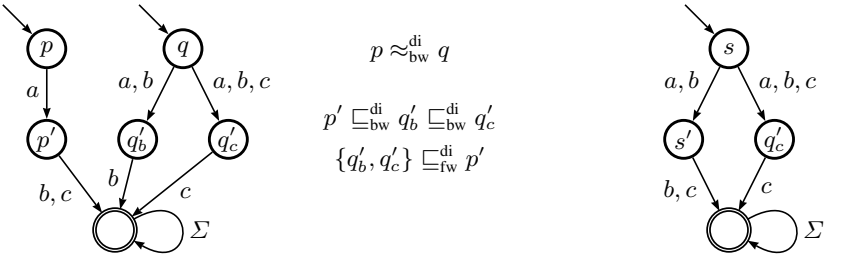
Let *direct proxy simulation*, written $\sqsubseteq^{\text{di}}_{\text{xy}}$, be defined as $\sqsubseteq^{\text{di}}_{\text{xy}} := [\tau_1(\sqsubseteq^{\text{di}}_{\text{bw}})]^{-1}$.

Theorem 7. *$\sqsubseteq^{di}_{xy}$ is a polynomial time GFQ preorder at least as coarse as $(\sqsubseteq^{di}_{bw})^{-1}$.*

Proxies vs mediators. Direct proxy simulation and mediated preorder [1] are in general incomparable. While proxy simulation is at least as coarse as backward direct simulation, mediated preorder is at least as coarse as *forward* direct simulation. (We have seen in Section 5 that this is somehow unavoidable, since one cannot hope to generalize simultaneously both forward and backward simulation.)

One notable difference between the two notions is that proxies are "dynamic", while mediators are "static": While Dupicator chooses the proxy only *after* Spoiler has selected her move, mediators are chosen uniformly w.r.t. Spoiler's move.

In Figure 3(a) we show a simple example where $\sqsubseteq^{\text{di}}_{\text{xy}}$ achieves greater reduction. Recall that mediated preorder M is always a subset of $\sqsubseteq^{\text{di}}_{\text{fw}} \circ (\sqsubseteq^{\text{di}}_{\text{bw}})^{-1}$ [1]. In the example, static mediators are just the trivial ones already present in forward simulation. Thus, $\sqsubseteq^{\text{di}}_{\text{fw}} \circ (\sqsubseteq^{\text{di}}_{\text{bw}})^{-1} = \sqsubseteq^{\text{di}}_{\text{fw}}$ and mediated preorder M collapses to forward simulation. On the other side, $p \approx^{\text{di}}_{\text{xy}} q$ and $p' \approx^{\text{di}}_{\text{xy}} q'_b$. Letting $s = [p, q]$ and $s' = [p', q'_b]$, we obtain the quotient in Figure 3(b).



(a) Original automaton (b) Quotient automaton

Fig. 3. Direct proxy simulation quotients

6.2 Delayed Proxy Simulation

Another difference between the mediated preorder approach [1] and the approach through proxies is that proxies directly enable a delayed simulation-like generalization (see Section 5.1). Again, we fix backward delayed simulation $\sqsubseteq^{\text{di}}_{\text{bw}}$ as a starting refinement, and we define *delayed proxy simulation* as $\sqsubseteq^{\text{de}}_{\text{xy}} := [\tau_1^{\text{de}}(\sqsubseteq^{\text{di}}_{\text{bw}})]^{-1}$.

Theorem 8. *$\sqsubseteq^{de}_{xy}$ is a polynomial time GFQ preorder.*

Notice that delayed proxy simulation is at least as coarse as direct proxy simulation. Moreover, quotients w.r.t. $\sqsubseteq^{\text{de}}_{\text{xy}}$ can be smaller than direct forward/backward/proxy and delayed simulation quotients by an arbitrary large factor. See Figure 4: Forward delayed simulation is just the identity, and no two states are direct backward or proxy simulation equivalent. But $q_i \sqsubseteq^{\text{di}}_{\text{bw}} s$ for any $0 < i \leq k-1$. This causes any two outer states q_i, q_j to be $\sqsubseteq^{\text{de}}_{\text{xy}}$-equivalent. Therefore, the $\sqsubseteq^{\text{de}}_{\text{xy}}$-quotient automaton has only 2 states.

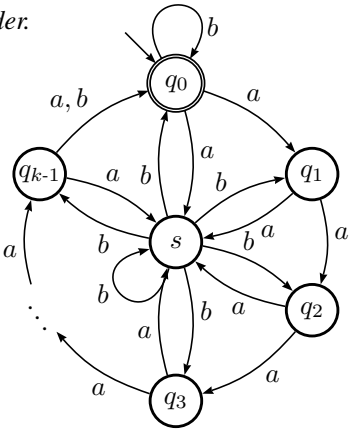


Fig. 4.

7 Conclusions and Future Work

We have proposed novel refinements for quotienting Büchi automata: fixed-word delayed simulation and direct/delayed proxy simulation. Each one has been shown to induce quotients smaller than previously known notions.

We outline a few directions for future work. First, we would like to study practical algorithms for computing fixed-word delayed simulation, and to devise efficient fragments thereof—one promising direction is to look at self-respecting fragments, which usually have lower complexity. Second, we would like to exploit the general correctness

argument developed in Section 4 in order to get efficient purely backward refinements (coarser than backward direct simulation). Finally, experiments on cases of practical interest are needed for an empirical evaluation of the proposed techniques.

Acknowledgment. We thank Richard Mayr and Patrick Totzke for helpful discussions, and two anonymous reviewers for their valuable feedback.

References

1. Abdulla, P., Chen, Y.-F., Holik, L., Vojnar, T.: Mediating for Reduction. In: FSTTCS, pp. 1–12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2009)
2. Aziz, A., Singhal, V., Swamy, G.M., Brayton, R.K.: Minimizing Interacting Finite State Machines. Technical Report UCB/ERL M93/68, UoC, Berkeley (1993)
3. Clemente, L.: Büchi Automata can have Smaller Quotients. Technical Report EDI-INF-RR-1399, LFCS, University of Edinburgh (April 2011)
4. Clemente, L., Mayr, R.: Multipebble Simulations for Alternating Automata. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 297–312. Springer, Heidelberg (2010)
5. Dill, D.L., Hu, A.J., Wont-Toi, H.: Checking for Language Inclusion Using Simulation Preorders. In: Larsen, K.G., Skou, A. (eds.) CAV 1991. LNCS, vol. 575, Springer, Heidelberg (1992)
6. Etessami, K.: A Hierarchy of Polynomial-Time Computable Simulations for Automata. In: Brim, L., Jančar, P., Křetínský, M., Kučera, A. (eds.) CONCUR 2002. LNCS, vol. 2421, pp. 131–144. Springer, Heidelberg (2002)
7. Etessami, K., Wilke, T., Schuller, R.A.: Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata. SIAM J. Comput. 34(5), 1159–1175 (2005)
8. Fritz, C., Wilke, T.: Simulation Relations for Alternating Büchi Automata. Theor. Comput. Sci. 338(1-3), 275–314 (2005)
9. Gramlich, G., Schnitger, G.: Minimizing NFA's and Regular Expressions. Journal of Computer and System Sciences 73(6), 908–923 (2007)
10. Gurumurthy, S., Bloem, R., Somenzi, F.: Fair Simulation Minimization. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 610–624. Springer, Heidelberg (2002)
11. Henzinger, T.A., Kupferman, O., Rajamani, S.K.: Fair Simulation. Information and Computation 173, 64–81 (2002)
12. Henzinger, T.A., Rajamani, S.K.: Fair Bisimulation. In: Graf, S. (ed.) TACAS 2000. LNCS, vol. 1785, pp. 299–314. Springer, Heidelberg (2000)
13. Jiang, T., Ravikumar, B.: Minimal NFA Problems are Hard. In: Leach Albert, J., Monien, B., Rodríguez-Artalejo, M. (eds.) ICALP 1991. LNCS, vol. 510, pp. 629–640. Springer, Heidelberg (1991)
14. Juvekar, S., Piterman, N.: Minimizing Generalized Büchi Automata. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 45–58. Springer, Heidelberg (2006)
15. Kupferman, O., Vardi, M.: Verification of Fair Transition Systems. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 372–382. Springer, Heidelberg (1996)
16. Kupferman, O., Vardi, M.: Weak Alternating Automata Are Not That Weak. ACM Trans. Comput. Logic 2, 408–429 (2001)
17. Lynch, N.A., Vaandrager, F.W.: Forward and Backward Simulations. Part I: Untimed Systems. Information and Computation 121(2), 214–233 (1995)

18. Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)
19. Raimi, R.S.: Environment Modeling and Efficient State Reachability Checking. PhD thesis, The University of Texas at Austin (1999)
20. Schewe, S.: Beyond Hyper-Minimisation—Minimising DBAs and DPAs is NP-Complete. In: FSTTCS. LIPIcs, vol. 8, pp. 400–411. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2010)
21. Somenzi, F., Bloem, R.: Efficient Büchi Automata from LTL Formulae. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 248–263. Springer, Heidelberg (2000)
22. Vardi, M.: Alternating Automata and Program Verification. In: van Leeuwen, J. (ed.) Computer Science Today. LNCS, vol. 1000, pp. 471–485. Springer, Heidelberg (1995)

Automata-Based CSL Model Checking

Lijun Zhang[1], David N. Jansen[2], Flemming Nielson[1], and Holger Hermanns[3]

[1] Technical University of Denmark, DTU Informatics, Denmark
[2] Radboud Universiteit, Model-based system design, Nijmegen, The Netherlands
[3] Saarland University, Computer Science, Saarbrücken, Germany

Abstract. For continuous-time Markov chains, the model-checking problem with respect to continuous-time stochastic logic (CSL) has been introduced and shown to be decidable by Aziz, Sanwal, Singhal and Brayton in 1996. The presented decision procedure, however, has exponential complexity. In this paper, we propose an effective approximation algorithm for full CSL.

The key to our method is the notion of *stratified CTMCs* with respect to the CSL property to be checked. We present a measure-preservation theorem allowing us to reduce the problem to a transient analysis on stratified CTMCs. The corresponding probability can then be approximated in polynomial time (using uniformization). This makes the present work the centerpiece of a broadly applicable full CSL model checker.

Recently, the decision algorithm by Aziz *et al.* was shown to be incorrect in general. In fact, it works only for stratified CTMCs. As an additional contribution, our measure-preservation theorem can be used to ensure the decidability for general CTMCs.

1 Introduction

Continuous-time Markov chains (CTMC) play an important role in performance evaluation of networked, distributed, and recently biological systems. The concept of formal verification for CTMCs was introduced by Aziz, Sanwal, Singhal and Brayton in 1996 [1,2]. Their seminal paper defined continuous-time stochastic logic (CSL) to specify properties over CTMCs. It showed that the model checking problem for CTMCs, which asks whether the CTMC satisfies a given CSL property, is decidable, using algebraic and transcendental number theory.

The characteristic construct of CSL is a probabilistic formula of the form $\mathcal{P}_{<p}(\varphi)$ where $p \in [0,1]$. φ is a path formula; more concretely, it is a *multiple until* formula $f_1\ U_{I_1}\ f_2\ U_{I_2}\ \ldots f_k$ where $k \geq 2$. The formula $\mathcal{P}_{<p}(\varphi)$ expresses a constraint on the probability to reach an f_k-state by passing only through (zero or more) f_1-, f_2-, ..., f_{k-1}-states in the given order. The key to solve the model checking problem is to approximate this probability $\Pr_s(\varphi)$ exact enough to decide whether it is $< p$. The decision procedure in [2] first decomposes the formula into (up to) exponentially many subformulas with suitable timing constraints. For each subformula, it then exploits properties of algebraic and transcendental numbers, but unfortunately lacks a practicable algorithm. In 2000, Baier *et al.* [4] presented an *approximate model checking algorithm* for

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 271–282, 2011.

the case $k = 2$. This algorithm is based on transient probability analysis for CTMCs. More precisely, it was shown that $\Pr_s(\varphi)$ can be approximated, up to a priori given precision ε, by a sum of transient probabilities in the CTMCs. Their algorithm then led to further development of approximation algorithms for infinite CTMCs and abstraction techniques. More importantly, several tools support approximate model checking, including PRISM [9], MRMC [11].

Approximate and effective model checking of full CSL with multiple until formulae ($k > 2$) is an open problem. This problem is gaining importance e. g. in the field of system biology, where one is interested in oscillatory behavior of CTMCs [5,13]. More precisely, if one intends to quantify the probability mass oscillating between high, medium and low concentrations (or numbers) of some species, a formula like *high* U_{I_1} *medium* U_{I_2} *low* U_{I_3} *medium* U_{I_4} *high* is needed, but this is not at hand with the current state of the art.

In this paper we propose an approximate algorithm for checking CSL with multiple until formulae. We introduce a subclass of *stratified CTMCs,* on which the approximation of $\Pr_s(\varphi)$ can be obtained by efficient transient analysis. Briefly, a CTMC is stratified with respect to $\varphi = f_1\ U_{I_1}\ f_2\ U_{I_2}\ \ldots f_k$, if the transitions of the CTMC respect some *order* given by the f_i. This specific order makes it possible to express $\Pr_s(\varphi)$ recursively: more precisely, it is the product of a transient vector and $\Pr_{s'}(\varphi')$, where φ' is a subformula of φ. Stratified CTMCs are the key element for our analysis: In a stratified CTMC, the problem reduces to a transient analysis. We extend the well-known result [4] for the case of binary until. Efficient implementations using *uniformization* [8] exist.

For a general CTMC, we present a measure-preserving transformation to a stratified CTMC. Our reduction is described using a *deterministic finite automaton* (DFA) over the alphabet $2^{\{f_1,\ldots,f_k\}}$. The DFA accepts the finite word $w = w_1 w_2 \ldots w_n$ if and only if the corresponding set of time-abstract paths in the CTMC contributes to $\Pr_s(\varphi)$, i. e., it respects the order of the f_i. The transformation does not require to construct the full DFA, but only the product of the CTMC and the DFA. We show that the product is a stratified CTMC, and moreover, the measure $\Pr_s(\varphi)$ is preserved. This product can be constructed in linear time and space. Thus our method will be useful as the centerpiece of a full CSL model checker equipped with multiple until formulae.

Recently, the decision algorithm by Aziz *et al.* was shown to produce erroneous results on some non-stratified CTMCs [10]. Still, their algorithm is correct on stratified CTMCs. As an additional contribution, our measure-preservation theorem ensures the decidability of CSL model checking for general CTMCs.

Full proofs are given in [16].

Overview of the article. Section 2 sets the ground for the paper. In Sect. 3 we introduce *stratified CTMCs* formally. The first main result is shown in Sect. 4: it constructs a DFA for an until formula, and then shows that the product is a stratified CTMC and the relevant measures are preserved. Section 5 discusses the computations in the product CTMC. A model checking algorithm is presented in Sect. 6. Section 7 discusses related work, and the paper is concluded in Sect. 8.

2 Preliminaries

2.1 Markov Chains

Definition 1. *A labeled discrete-time Markov chain (DTMC) is a tuple* $\mathcal{D} = (S, \mathbf{P}, L)$ *where* S *is a finite set of states,* $\mathbf{P} : S \times S \to [0,1]$ *is a probability matrix satisfying* $\sum_{s' \in S} \mathbf{P}(s,s') \in \{0,1\}$ *for all* $s \in S$*, and* $L : S \to 2^{AP}$ *is a labeling function.*

A labeled continuous-time Markov chain (CTMC) is a tuple $\mathcal{C} = (S, \mathbf{R}, L)$ *where* S *and* L *are defined as for DTMCs, and* $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ *is a rate matrix.*

For $A \subseteq S$, define $\mathbf{R}(s,A) := \sum_{s' \in A} \mathbf{R}(s,s')$, and let $E(s) := \mathbf{R}(s,S)$ denote the *exit rate* of s. A state s is called *absorbing* if $E(s) = 0$. If $\mathbf{R}(s,s') > 0$, we say that there is a transition from s to s'.

The transition probabilities in a CTMC are exponentially distributed over time. If s is the current state of the CTMC, the probability that some transition will be triggered within time t is $1 - e^{-E(s)t}$. Furthermore, if $\mathbf{R}(s,s') > 0$ for more than one state s', the probability to take a particular transition to s' is $\frac{\mathbf{R}(s,s')}{E(s)} \cdot \left(1 - e^{-E(s)t}\right)$. The labeling function L assigns to each state s a set of atomic propositions $L(s) \subseteq AP$ which are valid in s.

A CTMC $\mathcal{C} = (S, \mathbf{R}, L)$ (and also a DTMC) is usually equipped with an initial state $s_{\text{init}} \in S$ or, more generally, an initial distribution $\alpha_{\text{init}} : S \to [0,1]$ satisfying $\sum_{s \in S} \alpha_{\text{init}}(s) = 1$.

Transient probability. Starting with distribution α, the transient probability vector at time t, denoted by $\pi(\alpha, t)$, is the probability distribution over states at time t. If $t = 0$, we have $\pi(\alpha, 0)(s') = \alpha(s')$. For $t > 0$, the transient probability [14] is given by: $\pi(\alpha, t) = \pi(\alpha, 0)e^{\mathbf{Q}t}$ where $\mathbf{Q} := \mathbf{R} - \mathit{Diag}(E)$ is the *generator* matrix and $\mathit{Diag}(E)$ denotes the diagonal matrix with $\mathit{Diag}(E)(s,s) = E(s)$.

Paths and probabilistic measures. A (timed) infinite path is an infinite sequence $\sigma = s_0 t_0 s_1 t_1 \ldots$ satisfying $\mathbf{R}(s_i, s_{i+1}) > 0$, and $t_i \in \mathbb{R}_{>0}$ for all $i \geq 0$. For $i \in \mathbb{N}$, let $\sigma_S[i] = s_i$ denote the $(i+1)$-th state, and $\sigma_T[i] = t_i$ denote the time spent in s_i. For $t \in \mathbb{R}_{\geq 0}$, let $\sigma@t$ denote[1] the state s_i such that i is the smallest index with $t < \sum_{j=0}^{i} t_j$. A (timed) finite path is a finite sequence $\sigma = s_0 t_0 s_1 t_1 \ldots s_n$ satisfying $\mathbf{R}(s_i, s_{i+1}) > 0$ for all $0 \leq i < n$, and s_n is absorbing. For finite paths, the notations $\sigma_S[i]$ and $\sigma_T[i]$ are defined for $i \leq n$, and $\sigma_T[n]$ is defined to be ∞; otherwise, they are defined as above. Let $\mathit{Path}^{\mathcal{C}}$ denote the set of all (finite and infinite) paths, and $\mathit{Path}^{\mathcal{C}}(s)$ denote the subset of those paths starting from s.

Let $s_0, s_1, \ldots, s_k$ be states in S with $\mathbf{R}(s_i, s_{i+1}) > 0$ for all $0 \leq i < k$. Moreover, let $I_0, I_1, \ldots, I_{k-1}$ be nonempty intervals in $\mathbb{R}_{\geq 0}$. The *cylinder set* $\mathit{Cyl}(s_0, I_0, s_1, I_1, \ldots, s_{k-1}, I_{k-1}, s_k)$ is defined by:

$$\{\sigma \in \mathit{Path}^{\mathcal{C}} \mid \forall 0 \leq i \leq k.\ \sigma_S[i] = s_i \wedge \forall 0 \leq i < k.\ \sigma_T[i] \in I_i\}$$

[1] In [4], $\sigma@t$ is defined by requiring $t \leq \sum_{j=0}^{i} t_j$. However, the algorithm presented there follows our definition. The difference is discussed in [16].

Let $\mathcal{F}(Path^{\mathcal{C}})$ denote the smallest σ-algebra on $Path^{\mathcal{C}}$ containing all cylinder sets. For initial distribution $\alpha : S \to [0,1]$, a probability measure (denoted $\Pr^{\mathcal{C}}_{\alpha}$) on this σ-algebra is introduced as follows: $\Pr^{\mathcal{C}}_{\alpha}$ is the unique measure that satisfies: $\Pr^{\mathcal{C}}_{\alpha}(Cyl(s))$ equals $\alpha(s)$, and for $k > 0$,

$$\Pr^{\mathcal{C}}_{\alpha}(Cyl(s_0, I_0, \ldots, I_{k-1}, s_k)) = \Pr^{\mathcal{C}}_{\alpha}(Cyl(s_0, I_0, \ldots, s_{k-1})) \cdot \frac{\mathbf{R}(s_{k-1}, s_k)}{E(s_{k-1})} \cdot \eta(I_{k-1})$$

where $\eta(I_{k-1}) := \exp(-E(s_{k-1}) \inf I_{k-1}) - \exp(-E(s_{k-1}) \sup I_{k-1})$ is the probability to take a transition during time interval I_{k-1}. If $\alpha(s) = 1$ for some state $s \in S$, we sometimes simply write $\Pr^{\mathcal{C}}_{s}$ instead of $\Pr^{\mathcal{C}}_{\alpha}$. We omit the superscript $\mathcal{C}$ if it is clear from the context.

2.2 Deterministic Finite Automata

Definition 2. *A* deterministic finite automaton *is a tuple* $\mathcal{B} = (\Sigma, Q, q_{in}, \delta, F)$ *where* Σ *is a nonempty finite alphabet,* Q *is a finite set of states,* $q_{in} \in Q$ *is the initial state,* $\delta : Q \times \Sigma \nrightarrow Q$ *is the partial transition function, and* $F \subseteq Q$ *is the set of final states.*

We call a finite sequence $w = w_1 w_2 \ldots w_n$ over Σ a *word* over Σ. w induces at most one path $\sigma(w) = q_0 q_1 \ldots q_n$ in $\mathcal{B}$ where $q_0 = q_{in}$ and $q_i = \delta(q_{i-1}, w_i)$ for $i = 1, \ldots, n$. This word w, and also the corresponding path $\sigma(w)$, is *accepting* if $\sigma(w)$ exists and $q_n \in F$.

2.3 Continuous Stochastic Logic (CSL)

This section presents the branching-time temporal logic which allows us to specify properties over CTMCs. We consider the logic *Continuous Stochastic Logic* (CSL) introduced by Aziz *et al.* [2].

Let $I_1, I_2, \ldots$ be non-empty intervals on $\mathbb{R}_{\geq 0}$. Let $\trianglelefteq \in \{<, \leq, \geq, >\}$, $0 \leq p \leq 1$, and $k \geq 2$. The syntax of the logic CSL is defined as follows:

$$\Phi := true \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathcal{P}_{\trianglelefteq p}(\varphi)$$
$$\varphi := \Phi_1 \, U_{I_1} \, \Phi_2 \, U_{I_2} \ldots \Phi_k$$

where $a \in AP$ is an atomic proposition. We use the abbreviation $\Diamond_I \Phi = true \, U_I \, \Phi$. The syntax of CSL consists of state formulae and path formulae: we use $\Phi, \Phi_1, \Psi, \Psi_1, \ldots$ for state formulae and $\varphi, \varphi_1, \psi, \psi_1, \ldots$ for path formulae.

Let $\mathcal{C} = (S, \mathbf{R}, L)$ be a CTMC with $s \in S$. The semantics of CSL state formulae is standard: $s \models true$ for all $s \in S$, $s \models a$ iff $a \in L(s)$, $s \models \neg\Phi$ iff $s \not\models \Phi$, $s \models \Phi \wedge \Psi$ iff $s \models \Phi$ and $s \models \Psi$. For probabilistic formulae, we have:

$$s \models \mathcal{P}_{\trianglelefteq p}(\varphi) \text{ iff } \Pr_s\{\sigma \in Path \mid \sigma \models \varphi\} \trianglelefteq p$$

where $\Pr_s\{\sigma \in Path \mid \sigma \models \varphi\}$, or $\Pr_s(\varphi)$ for short, denotes the probability measure of the set of all paths which start with s and satisfy φ.

The satisfaction relation for CSL path formulae is defined as follows: Let σ be a path, and let $\varphi = \Phi_1\ U_{I_1}\ \Phi_2\ U_{I_2}\ \ldots \Phi_k$ be a path formula. Then, $\sigma \models \varphi$ if and only if there exist real numbers $0 \leq t_1 \leq t_2 \leq \ldots \leq t_{k-1}$ such that $\sigma@t_{k-1} \models \Phi_k$, and for each integer $0 < i < k$ we have $(t_i \in I_i) \wedge (\forall t' \in [t_{i-1}, t_i))(\sigma@t' \models \Phi_i)$, where t_0 is defined to be 0 for notational convenience.

Let $\varphi = \Phi_1\ U_{I_1}\ \Phi_2\ U_{I_2}\ \ldots \Phi_k$ be a CSL path formula with $a_i = \inf I_i$ and $b_i = \sup I_i$ (which can be ∞). We say that φ is *well-formed* if I_i is not empty (for all i) and $a_1 \leq a_2 \leq \ldots a_{k-1}$ and $b_1 \leq b_2 \leq \ldots b_{k-1}$. Obviously, if some I_i is empty, φ is false. If $a_i < a_{i-1}$ or $b_i > b_{i+1}$, replace a_i by $\max\{a_1, \ldots, a_i\}$ and b_i by $\min\{b_i, \ldots, b_{k-1}\}$ to get an equivalent well-formed formula. Therefore, we can assume w. l. o. g. that path formulae are well-formed.

3 Stratified CTMCs

The main challenge of model checking is the approximation of the probability $\Pr_s(\varphi)$. We now introduce the class of *stratified* CTMCs with respect to a path formula φ containing only atomic propositions as subformulas. Stratification is the key for the computation of $\Pr_s(\varphi)$. Later, we shall see that the definition is easily generalized to formulas containing more complex subformulas.

Let $\mathcal{C} = (S, \mathbf{R}, L)$ be a CTMC. Let $\varphi = f_1\ U_{I_1}\ f_2\ U_{I_2}\ \ldots f_k$ be a CSL path formula, with $F := \{f_1, f_2, \ldots, f_k\}$ the set of atomic propositions appearing in φ. Moreover, we let $\sqsubseteq$ be an order on F such that $f_i \sqsubseteq f_j$ iff $i \leq j$. For a state s, if the set $\{1 \leq i \leq k \mid f_i \in L(s)\}$ is not empty, we let $f^s_{\min} := f_j$ with $j = \min\{1 \leq i \leq k \mid f_i \in L(s)\}$. If such f_j does not exist, we define $f^s_{\min} := \bot$.

Definition 3 (Stratified CTMC). *We say that $\mathcal{C}$ is* stratified *with respect to φ iff for all s_1, s_2, it holds that:*

- *If $f^{s_1}_{\min} = \bot$ or $f^{s_1}_{\min} = f_k$, then $\mathbf{R}(s_1, s_2) = 0$,*
- *Otherwise (i. e., $f^{s_1}_{\min} \neq \bot$ and $f^{s_1}_{\min} \neq f_k$), if $\mathbf{R}(s_1, s_2) > 0$ and $f^{s_2}_{\min} \neq \bot$, then $f^{s_1}_{\min} \sqsubseteq f^{s_2}_{\min}$.*

A state s with $f^s_{\min} = \bot$ is referred to as *bad state*, and with $f^s_{\min} = f_k$ as *good state*. (Note that there may be other states satisfying f_k as well.) Both good and bad states are absorbing. The intuition behind Def. 3 is that paths reaching bad states will not satisfy φ, while those reaching good states (or other f_k-states) may satisfy φ (provided the time constraints are also satisfied).

Example 1. Consider the CTMC in Fig. 1 and $\varphi := f_1\ U_{[0,2]}\ f_2\ U_{[2,4]}\ f_3\ U_{[2,4]}\ f_4\ U_{[3,5]}\ f_5$. $\mathcal{C}$ is not stratified with respect to φ: we have $\mathbf{R}(s_2, s_1) > 0$, however, $f^{s_2}_{\min} = f_4 \not\sqsubseteq f_1 = f^{s_1}_{\min}$. Deleting this edge and the transition out of s_4 would result in a stratified CTMC with respect to φ. □

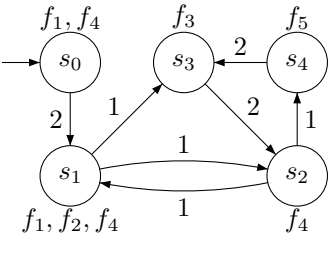


Fig. 1. A non-stratified CTMC

3.1 Usefulness of Stratification

We have introduced the notion of stratified CMTCs, which is the key to present an efficient approximation algorithm. The essential idea is that we can reduce the problem to a similar one on stratified CTMCs: details shall be presented later.

It is very interesting to note that our notion of stratified CTMCs solves a semantical problem, which we recently pointed out in [10]. Very briefly, Aziz *et al.* [2] gave an algorithm that did not use the t_i (in the semantics of until formulae) explicitly, which led to wrong results for non-stratified CTMCs. In Example 1, their algorithm treated paths containing $s_2 \to s_1 \to s_3$ (with suitable timing) as satisfying the formula φ. On the other hand, it treated correct paths that leave s_4 before time 5 as not satisfying φ.

4 Product CTMC

Given a CTMC and a CSL path formula φ, in this section we construct a stratified CTMC with respect to φ that has the same probabilities of satisfying φ. We first construct a deterministic finite automaton for φ in Subsect. 4.1. Then, in Subsect. 4.2 we build a product CTMC with the desired property.

4.1 Automaton for a CSL Formula

For a formula $\varphi = f_1\ U_{I_1}\ f_2\ U_{I_2}\ \ldots f_k$, we first construct a simple deterministic finite automaton that describes the required order of f_1-, f_2-, ..., f_k-states.

Definition 4 (Formula automaton). *Let $\varphi = f_1\ U_{I_1}\ f_2\ U_{I_2}\ \ldots f_k$ be a CSL path formula. Then, the* formula automaton $\mathcal{B}_\varphi = (\Sigma, Q, q_{in}, \delta, F)$ *is defined by:* $\Sigma = 2^{\{f_1,\ldots,f_k\}}$, $Q = \{q_1, q_2, \ldots, q_{k-1}, q_k, \bot\}$ *with* $q_{in} = q_1$ *and* $F = \{q_1, \ldots, q_k\}$. *For $a \in \Sigma$, the transition relation δ is defined as follows:*

1. $\delta(q_i, a) = q_j$ *if* $i < k$, $i \le j \le k$, $f_i, f_{i+1} \ldots f_{j-1} \notin a$, $f_j \in a$;
2. $\delta(q_i, a) = \bot$ *if* $i < k$ *and the above clause does not apply;*
3. $\bot$ *and* q_k *are absorbing.*

The words accepted by $\mathcal{B}_\varphi$ are finite traces $w \in \Sigma^*$, such that they can be extended to a trace $ww' \in \Sigma^\omega$ that satisfies the time-abstract (LTL) formula of the form $f_1\ U\ (f_2\ U\ (\ldots (f_{k-1}\ U\ f_k)\ldots))$.

We mention that in the automaton-based LTL model checking algorithm [15], the size of a *nondeterministic Büchi automaton* is *exponential* in the size of the formula, and determinisation requires another exponentiation [12]. The constructed finite automaton $\mathcal{B}_\varphi$ for this special class of formulae is deterministic, the number of states is linear in k. The number of transitions is $(k-1)2^k$; however, as we will see later, the product can be constructed in time (and size) linear in the size of the CTMC and in k.

Example 2. In Fig. 2 the formula automaton for $k = 4$ is illustrated. The initial state is q_1, final states are marked with a double circle. The transition labels indicate which subsets of AP are acceptable. For example, we have $\delta(q_1, \{f_1\}) = \delta(q_1, \{f_1, f_2\}) = q_1$, as both sets $\models f_1$. □

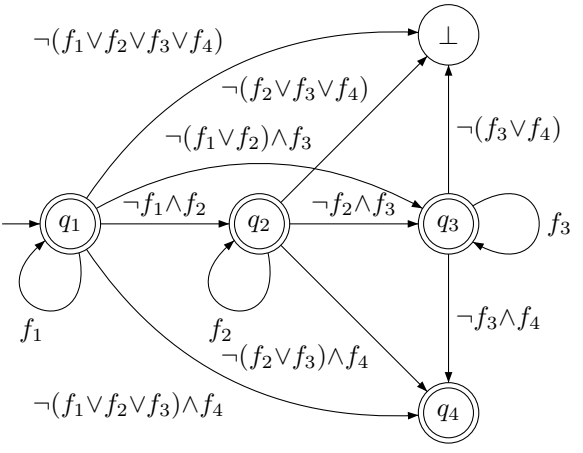


Fig. 2. $\mathcal{B}_\varphi$ for $\varphi = f_1\ U\ f_2\ U\ f_3\ U\ f_4$

4.2 Product CTMC

Below we define the *product CTMC* of $\mathcal{C}$ and the formula automaton $\mathcal{B}_\varphi$:

Definition 5 (Product CTMC). *Let $\mathcal{C} = (S, \mathbf{R}, L)$ be a CTMC, and $\varphi = f_1\ U_{I_1}\ f_2\ U_{I_2} \ldots f_k$ a CSL formula. Let $\mathcal{B}_\varphi$ be as constructed above. The product $\mathcal{C} \times \mathcal{B}_\varphi$ is a CTMC $(S', \mathbf{R}', L')$ where:*

1. *$S' = S \times Q$,*
2. *$\mathbf{R}'((s, q_i), (s', q'))$ equals $\mathbf{R}(s, s')$ if $s \models_{\mathcal{C}} \bigvee_{j=i}^{k-1} f_j$ and $q' = \delta(q_i, L(s'))$, and equals 0 otherwise,*
3. *the labeling function is defined by:*
   - *$L'(s, q_i) = L(s) \setminus \{f_1, f_2, \ldots, f_{i-1}\}$ for $1 \leq i \leq k$,*
   - *$L'(s, \bot) = \emptyset$.*
4. *Given an initial distribution $\alpha : S \to [0, 1]$ of $\mathcal{C}$, the initial distribution of the product $\alpha' : S \times Q \to [0, 1]$ is defined by: $\alpha'(s, q)$ equals $\alpha(s)$ if $q = \delta(q_{in}, L(s))$, and equals 0 otherwise.*

The product CTMC contains two kinds of absorbing states. In general, states (s, q) with $s \not\models \bigvee_{i=1}^{k} f_i$ are absorbing in the product, as well as states whose propositions do not fit the current phase. These two kinds of states can be considered bad states. On the other hand, good states of the form (s, q_k) are also absorbing. The behavior after absorbing states is irrelevant for determining the probability to satisfy φ.

Example 3. Consider the CTMC in Fig. 1, and consider the path formula $\varphi_1 := f_1\ U_{[0,2)}\ f_2\ U_{[0,2)}\ f_3\ U_{[0,2)}\ f_4\ U_{[0,2)}\ f_5$. The path $\sigma_1 := s_0 s_1 s_3 s_2 s_4$ does, if s_4 is reached before time 2, satisfy φ_1; however, the path $\sigma_2 := s_0 s_1 s_2 s_1 s_3 s_2 s_4$ does not. Therefore, $\Pr_{s_0}(\varphi_1)$ cannot be computed by standard probabilistic reachability analysis. The product of this CTMC with $\mathcal{B}_{\varphi_1}$ is the CTMC depicted on the left of Fig. 3. State (s_4, q_5) is a *good state* – paths reaching this state before time 2 correspond to paths satisfying φ_1 in Fig. 1 –, while $(s_3, \bot)$ is a *bad state.*

For the same CTMC in Fig. 1, consider the path formula $\varphi_2 := f_1\ U_{[1,3)}\ f_2\ U_{[1,3)}\ f_3\ U_{[1,3)}\ f_4$. The product CTMC $\mathcal{C} \times \mathcal{B}_{\varphi_2}$ is depicted on the right of

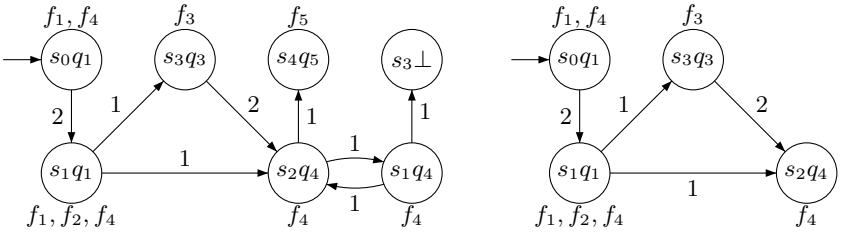


Fig. 3. The reachable part of the product CTMC $\mathcal{C} \times \mathcal{B}_{\varphi_1}$ (left) and $\mathcal{C} \times \mathcal{B}_{\varphi_2}$ (right)

Fig. 3. It is easy to check that the product CTMC is stratified with respect to φ_2. The absorbing state (s_2, q_4) is a *good state.* □

For a CTMC $\mathcal{C}$ and a state s, we use $\mathcal{C}|_s = (S', \mathbf{R}', L')$ to denote the *sub-CTMC* reachable from s, i.e., $S' \subseteq S$ is the states reachable from s, $\mathbf{R}'$ and L' are functions restricted to $S' \times S'$ and S', respectively.

Theorem 1 (Measure-preservation theorem). *Let $\mathcal{C} = (S, \mathbf{R}, L)$ be a CTMC, and $\varphi = f_1\ U_{I_1}\ f_2\ U_{I_2}\ \ldots f_k$ a CSL path formula. Let $\mathcal{B}_\varphi$ denote the formula automaton. For $s \in S$, let $s_B = \delta(q_{in}, L(s))$. Then, it holds:*

1. *$\mathcal{C} \times \mathcal{B}_\varphi|_{s_B}$ is stratified with respect to φ;*
2. $\Pr_s^{\mathcal{C}}(\varphi) = \Pr_{s_B}^{\mathcal{C} \times \mathcal{B}_\varphi|_{s_B}}(\varphi) = \Pr_{s_B}^{\mathcal{C} \times \mathcal{B}_\varphi}(\varphi)$.

Size of the Product CTMC. The number of states is linear in k, and the number of transitions in $\mathcal{B}_\varphi$ is exponential in k. It is interesting to notice, however, that the size of the product is only in $\mathcal{O}(mk)$. To see this, first note that the number of states in the product is nk, where n denotes the number of states of the CTMC. Since the DFA is deterministic, for each state (s, q), the outgoing transitions exactly correspond to the outgoing transitions from s, implying that the total number of transitions is in $\mathcal{O}(mk)$.

5 Computing $\Pr_\alpha(\varphi)$

This section aims at computing the probability $\Pr_\alpha(\varphi)$ starting from an arbitrary initial distribution α. We fix a CSL path formula $\varphi = f_1\ U_{I_1}\ f_2\ U_{I_2}\ \ldots f_k$, together with a stratified CTMC $\mathcal{C} = (S, \mathbf{R}, L)$ with respect to φ. We first prove a technical lemma.

Lemma 2 (Closure of Intervals). *Let $s \in S$. Assume given two intervals I, J such that $\inf I = \inf J$ and $\sup I = \sup J$. Then, it holds:*

1. *If $0 \in I \Leftrightarrow 0 \in J$, then $s \models \mathcal{P}_{\trianglelefteq p}(\Phi\ U_I\ \Psi)$ iff $s \models \mathcal{P}_{\trianglelefteq p}(\Phi\ U_J\ \Psi)$, for $\trianglelefteq \in \{<, \leq, \geq, >\}$.*
2. *Otherwise, assume w. l. o. g. $0 \in I$ and $0 \notin J$. Then, $s \models \mathcal{P}_{\trianglerighteq p}(\Phi\ U_I\ \Psi) \wedge \Phi$ iff $s \models \mathcal{P}_{\trianglerighteq p}(\Phi\ U_J\ \Psi)$, for $\trianglerighteq \in \{\geq, >\}$. Similarly, $s \models \mathcal{P}_{\trianglelefteq p}(\Phi\ U_I\ \Psi) \vee \neg\Phi$ iff $s \models \mathcal{P}_{\trianglelefteq p}(\Phi\ U_J\ \Psi)$, for $\trianglelefteq \in \{<, \leq\}$.*

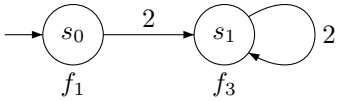


Fig. 4. Does $f_2\ U_{(0,1]}\ f_1$ hold?

The lemma follows immediately from the definition of the measure of cylinder set. To see why we have to treat the case $\inf I = 0$ separately (not distinguished in [4]), assume that $\varphi = f_2\ U_{(0,1]}\ f_1$ and consider the CTMC depicted in Fig. 4: obviously we have $s_0 \models \mathcal{P}_{\leq 0}(\varphi)$ as $s_0 \not\models f_2$. However, $s_0 \models \mathcal{P}_{\geq 1}(f_2\ U_{[0,1]}\ f_1)$ as s_0 satisfies f_1 directly. In this case, the first formula is equivalent to $\mathcal{P}_{\leq 0}(f_2\ U_{[0,1)}\ f_1) \vee \neg f_2$. This lemma allows us to use left-closed intervals $[a, b)$ only in the remainder of the article.

For an interval I and x with $x < \sup I$, we let $I \ominus x$ denote the set $\{t - x \mid t \in I \wedge t \geq x\}$. For example we have $[3, 8) \ominus 5 = [0, 3)$. Then, for $\varphi = f_1\ U_{I_1}\ f_2\ U_{I_2}\ \ldots f_k$ and $x < \sup I_1$, we let $\varphi \ominus x$ denote the formula $f_1\ U_{I_1 \ominus x}\ f_2\ U_{I_2 \ominus x}\ \ldots f_k$. For $1 \leq j' \leq j \leq k$, define $f_{j' \ldots j} := \bigvee_{i=j'}^{j} f_i$; for $1 \leq j < k$, define $\varphi_j := f_j\ U_{I_j}\ f_{j+1}\ U_{I_{j+1}}\ \ldots f_k$. For Φ, we denote by $\mathcal{C}[\Phi]$ the CTMC obtained by $\mathcal{C}$ by making states satisfying Φ absorbing. Moreover, let $\mathbf{I}_\Phi$ denote the indicator matrix defined by: $\mathbf{I}_\Phi(s, s) = 1$ if $s \models \Phi$, and $\mathbf{I}_\Phi(s, s') = 0$ otherwise.

Theorem 3. *Let $\varphi = f_1\ U_{I_1}\ f_2\ U_{I_2}\ \ldots f_k$ be a CSL path formula, and assume all $I_i = [a_i, b_i)$ are left-closed. Let $\mathcal{C} = (S, \mathbf{R}, L)$ be a stratified CTMC with respect to φ. We write the vector $(\Pr_s^{\mathcal{C}}(\psi))_{s \in S}$ as $\Pr_{(\cdot)}^{\mathcal{C}}(\psi)$.*

1. *Assume $0 < a_1$. Then,*

$$\Pr_\alpha^{\mathcal{C}}(\varphi) = \pi^{\mathcal{C}[\neg f_1]}(\alpha, a_1) \cdot \mathbf{I}_{f_1} \cdot \Pr_{(\cdot)}^{\mathcal{C}}(\varphi \ominus a_1) \tag{1}$$

2. *Assume $0 = a_1 = \ldots = a_{j-1} < a_j < b_1$ for some $j \in \{2, \ldots, k-1\}$. Then,*

$$\Pr_\alpha^{\mathcal{C}}(\varphi) = \pi^{\mathcal{C}[\neg f_{1 \ldots j}]}(\alpha, a_j) \cdot \mathbf{I}_{f_{1 \ldots j}} \cdot \Pr_{(\cdot)}^{\mathcal{C}}(\varphi \ominus a_j) \tag{2}$$

3. *To simplify notation, let $a_k := \infty$ and $\Pr_{(\cdot)}^{\mathcal{C}[\neg f_{k \ldots k-1}]}(\varphi_k \ominus b_1) := (1, \ldots, 1)^{\mathrm{T}}$. Assume $0 = a_1 = \ldots = a_{j-1} < b_1 \leq a_j$ for some $j \in \{2, \ldots, k\}$. Let $j' \leq j$ be the largest integer such that $b_1 = \ldots = b_{j'-1}$. Then,*

$$\Pr_\alpha^{\mathcal{C}}(\varphi) = \pi^{\mathcal{C}[\neg f_{1 \ldots j}]}(\alpha, b_1) \cdot \mathbf{I}_{f_{j' \ldots j}} \cdot \Pr_{(\cdot)}^{\mathcal{C}[\neg f_{j' \ldots k-1}]}(\varphi_{j'} \ominus b_1) \tag{3}$$

If $b_1 = \infty$, we replace $\pi^{\mathcal{C}[\neg f_{1 \ldots j}]}(\alpha, b_1)$ in this equation by the obvious limit.

6 Model Checking Algorithm

Let $\mathcal{C} = (S, \mathbf{R}, L)$ be an CTMC with $s \in S$, and Φ be a CSL formula. The model checking problem is to check whether $s \models \Phi$. The standard algorithm to solve CTL-like model checking problems recursively computes the sets $Sat(\Psi)$ for all state subformulae Ψ of Φ. For CSL, the cases where Ψ is an atomic proposition, a negation or a conjunction are given by: $Sat(a) = \{s \in S \mid a \in L(s)\}$, $Sat(\neg\Psi_1) = S \backslash Sat(\Psi_1)$ and $Sat(\Psi_1 \wedge \Psi_2) = Sat(\Psi_1) \cap Sat(\Psi_2)$.

The case that Ψ is the probabilistic operator is the challenging part. Let $\Psi = \mathcal{P}_{\trianglelefteq p}(\varphi)$ with $\varphi = \Psi_1\ U_{I_1}\ \Psi_2\ U_{I_2}\ \ldots \Psi_k$. By the semantics, checking Ψ is equivalent

to check whether $\Pr_s(\varphi)$ meets the bound $\trianglelefteq p$, i. e., whether $\Pr_s(\varphi) \trianglelefteq p$. Assume that the sets $Sat(\Psi_i)$ have been calculated recursively. We replace $\Psi_1, \ldots, \Psi_k$ by fresh atomic propositions $f_1, \ldots, f_k$ and extend the label of state s by f_i if $s \in Sat(\Psi_i)$. The so obtained path formula is $\psi := f_1\ U_{I_1}\ f_2\ U_{I_2}\ \ldots f_k$, and obviously we have $\Pr_s(\varphi) = \Pr_s(\psi)$.

The steps needed to approximate $\Pr_s(\psi)$ are: (i) Construct the formula automaton $\mathcal{B}_\psi$. (ii) Build the product $\mathcal{C} \times \mathcal{B}_\psi$, which by Thm. 1 is a stratified CTMC with respect to ψ. (iii) Apply Thm. 3 repeatedly to compute $\Pr_s(\psi)$. Fortunately, it is possible to combine steps (i) and (ii) and construct the product immediately, without having to construct the full automaton $\mathcal{B}_\psi$, because we use the product construction only for very specific automata.

Lemma 4 (Complexity). *Let m denote the number of transitions of $\mathcal{C}$ and $\lambda \in \mathbb{R}_{>0}$ the* uniformization rate *satisfying $\lambda = \max_{s \in S}(E(s) - \mathbf{R}(s,s))$. For each formula $\varphi = f_1\ U_{I_1}\ f_2\ U_{I_2}\ \ldots f_k$, the probability $\Pr_{s_B}^{\mathcal{C} \times \mathcal{B}_\varphi}(\varphi)$ can be approximated:*

- *in time in $\mathcal{O}(m\lambda b \cdot k)$ if $b = \sup I_{k-1}$ is finite,*
- *in time in $\mathcal{O}(m\lambda b \cdot k + (|S|k)^3)$ if $\sup I_{k-1}$ is infinite, where $|S|$ is the number of states in $\mathcal{C}$ and $b = \max(\{\inf I_{k-1}\} \cup \{\sup I_i | 1 \leq i < k\} \setminus \{\infty\})$.*

The space complexity is in $\mathcal{O}(mk)$.

Thus, with the notion of stratified CTMC, we achieve polynomial complexity. Our algorithm therefore improves the work of [2], where only multiple until formulae with suitable timing constraints can be checked polynomially. In the worst case, [2] has to decompose a CSL formula into $\mathcal{O}(k^k)$ formulae with suitable timing, thus resulting in an overall time complexity of $\mathcal{O}(m\lambda b \cdot k^{k+1})$ or $\mathcal{O}((m\lambda bk + (|S|k)^3) \cdot k^k)$, respectively.

7 Related Work

The logic CSL was first proposed in [1], in which it is shown to be decidable. Our paper gives a practical solution: it shows that it can be approximated efficiently. For the case of single until path formula, Baier *et al.* [4] have presented an approximate algorithm for the model checking problem. Their method can be considered a special case of our approach.

Baier *et al.* [3] defined a logic asCSL that uses so-called programs as path formulas, i. e. regular expressions over state formulas and actions. Programs can express nested until formulas of the form $\varphi_1\ U_{[0,b)}\ \varphi_2\ U_{[0,b)} \cdots U_{[0,b)}\ \varphi_k$ because asCSL cannot restrict the duration of individual program phases. The model checking algorithm translates the program to an automaton almost equal to the one in Fig. 2. Our work generalizes the method to nested until formulas with multiple time bounds.

More recently, Donatelli *et al.* [7] have extended CSL such that path properties can be expressed via a *deterministic timed automata (DTA) with a single clock.* Chen *et al.* [6] take this trajectory further and consider DTA specifications with multiple clocks as well.

In principle, one can translate a nested until formula to a DTA with a single clock. Its basic structure would look like similar to Fig. 2, but Donatelli's and Chen's DTAs also include all timing information and would have a size in $\mathcal{O}(k^2)$. To check whether a CTMC satisfies a DTA specification, they build the product of the two, apply the region construction, and then solve a system of integral equations. Chen's method, applied directly to our specifications, would amount to a complexity in $\mathcal{O}(k^4|S|\lambda c + k^9|S|^3)$, where c is the largest difference between time constraints (roughly comparable to b in Lemma 4). Note that our algorithm has only a complexity in $\mathcal{O}(m\lambda bk)$ if $b = \sup I_{k-1} < \infty$ or $\mathcal{O}(m\lambda bk + (|S|k)^3)$ otherwise.

8 Conclusion

In this paper we have proposed an effective approximation algorithm for CSL with a multiple until operator. We believe that it is the centerpiece of a broadly applicable full CSL model checker.

The technique we have developed in this paper can also be applied to a subclass of PCTL* formulae. Let $\varphi = f_1\ U_{I_2}\ f_2\ U_{I_2}\ \ldots f_k$ be a CSL path formula. As we have seen in the paper, in case of $I = [0, \infty)$, our multiple until formula $f_1\ U_I\ f_2\ U_I\ \ldots f_k$ corresponds to the LTL formula $f_1\ U\ (f_2\ U\ (\ldots(f_{k-1}\ U\ f_k)\ldots))$. In general, φ is similar to a *step-bounded* LTL formula $\varphi = f_1\ U_{[i_1,j_1]}\ f_2\ U_{[i_2,j_2]}\ \ldots f_k$ with $i_1, j_1, \ldots$ integers specifying the step bounds. In the automaton-based model checking approach for DTMCs [15], such step-bounded until LTL formulae can be first transformed into nested *next-state* formulae, for example we have: $f_1\ U_{[2,3]}\ f_2 = f_1 \wedge X(f_1 \wedge X(f_2 \vee (f_1 \wedge X(f_2))))$.

Obviously, the length of the induced LTL formula is at least j_{k-1}, and a deterministic Büchi automaton is again double exponential in j_{k-1}, thus renders the automaton-based approach unattractive. The approach we have established in this paper can be adapted slightly to handle this kind of formulae in complexity linear in j_{k-1} (assuming $j_{k-1} < \infty$).

We conclude the paper by noting the connection of our DFA-based approach with the classical *Büchi-automaton*-based LTL model checking algorithm by Vardi and Wolper [15]. Büchi automata are shown to have exactly the same expressive power as LTL formula, and can be constructed in exponential time with respect to the length of the LTL formula. Then, model checking LTL can be reduced to automata-theoretic questions in the product. Instead of Büchi automata accepting infinite runs, we only need DFAs, which is due to the simple form of the multiple until formula, which does not encompass full LTL expressivity. This simplification, moreover, allows us to get a DFA whose number of states is only linear in the length of the CSL formula, and the size of the product automaton is then linear in both the size of the CTMC and the length of the CSL formula.

Acknowledgement. Lijun Zhang and Flemming Nielson are partially supported by MT-LAB, a VKR Centre of Excellence. David N. Jansen and Holger Hermanns are partially supported by DFG/NWO Bilateral Research Programme ROCKS and by the European Community's Seventh Framework Programme under grant agreement no. ICT-214755 (QUASIMODO).

References

1. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Verifying continuous time Markov chains. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 269–276. Springer, Heidelberg (1996)
2. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continous-time Markov chains. ACM Trans. Comput. Log. 1(1), 162–170 (2000)
3. Baier, C., Cloth, L., Haverkort, B.R., Kuntz, M., Siegle, M.: Model checking Markov chains with actions and state labels. IEEE Trans. Softw. Eng. 33(4), 209–224 (2007)
4. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. IEEE Trans. Softw. Eng. 29(6), 524–541 (2003)
5. Ballarini, P., Mardare, R., Mura, I.: Analysing biochemical oscillation through probabilistic model checking. Electr. Notes Theor. Comp. Sc. 229(1), 3–19 (2009)
6. Chen, T., Han, T., Katoen, J.P., Mereacre, A.: Quantitative model checking of continuous-time Markov chains against timed automata specifications. In: Twenty-fourth Annual IEEE Symposium on Logic in Computer Science, pp. 309–318. IEEE Comp. Soc., Los Alamitos (2009)
7. Donatelli, S., Haddad, S., Sproston, J.: Model checking timed and stochastic properties with $\mathrm{CSL}^{\mathrm{TA}}$. IEEE Trans. Software Eng. 35(2), 224–240 (2009)
8. Grassmann, W.K.: Finding transient solutions in Markovian event systems through randomization. In: Stewart, W.J. (ed.) Numerical Solution of Markov Chains. Probability, Pure and Applied, vol. 8, pp. 357–371. Marcel Dekker, New York (1991)
9. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: a tool for automatic verification of probabilistic systems. In: Hermanns, H. (ed.) TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
10. Jansen, D.N.: Erratum to: Model-checking continuous-time Markov chains by Aziz et al. (February 2011), http://arxiv.org/abs/1102.2079v1
11. Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Performance Evaluation 68(2), 90–104 (2011)
12. Safra, S.: On the complexity of ω-automata. In: 29th Ann. Symp. on Foundations of Computer Science, pp. 319–327. IEEE Comp. Soc., Los Alamitos (1988)
13. Spieler, D.: Model checking of oscillatory and noisy periodic behavior in Markovian population models. Master's thesis, Saarland University, Saarbrücken (2009), http://alma.cs.uni-sb.de/data/david/mt.pdf
14. Stewart, W.J.: Introduction to the numerical solution of Markov chains. Princeton Univ. Pr., Princeton (1994)
15. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Symposium on Logic in Computer Science, pp. 332–345. IEEE Comp. Soc., Los Alamitos (1986)
16. Zhang, L., Jansen, D.N., Nielson, F., Hermanns, H.: Automata-based CSL model checking (April 2011), http://arxiv.org/abs/1104.4983

A Progress Measure for Explicit-State Probabilistic Model-Checkers*

Xin Zhang and Franck van Breugel

DisCoVeri Group, Department of Computer Science and Engineering
York University, 4700 Keele Street, Toronto, M3J 1P3, Canada

Abstract. Verification of the source code of a probabilistic system by means of an explicit-state model-checker is challenging. In most cases, the probabilistic model-checker will run out of memory due to the infamous state space explosion problem. As far as we know, we are the first to introduce the notion of a progress measure for such a model-checker. The progress measure returns a number in the interval [0, 1]. This number captures the amount of progress the model-checker has made towards verifying a particular linear-time property. The larger the number, the more progress the model-checker has made. We prove that the progress measure provides a lower bound of the measure of the set of execution paths that satisfy the property. We also show how to compute the progress measure for checking a particular class of linear-time properties, namely invariants.

Keywords: probabilistic model-checking, progress measure, linear-time property, invariant.

1 Introduction

Model-checkers such as PRISM [5] and MRMC [3] have been successfully exploited to check properties of probabilistic systems. Such verification tools consider a model of the system, rather than the actual source code of the system. A model is usually simpler than the source code and, hence, the model is generally easier to verify. However, the model abstracts from certain details and those details are not considered in the verification effort and, hence, violations of certain properties might not be detected. But such violations might well be detected by model-checkers that consider the source code of the system.

Whereas a tool that checks properties of a model is usually exploited to find errors in algorithms, a tool that considers the source code is generally used to detect coding errors. Hence, both types of tool play their role in the verification process. In this paper, we consider the applicability of model-checkers to verify the source code of probabilistic systems. In particular, we focus on explicit-state model-checkers, that is, model-checkers in which the states of the systems are represented explicitly.

* This research is supported by NSERC.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 283–294, 2011.

One may wonder if an explicit-state model-checker, such as Java PathFinder (JPF for short) [9], is suitable for verifying the source code of probabilistic systems. Consider the following Java snippet.

```
Random random = new Random();
long value = 0;
while (random.nextBoolean())
   value++;
```

The variable `value` is initialized to zero and is incremented as long as the random Boolean value returned by the method invocation `random.nextBoolean()` returns true. The above snippet gives rise to a huge number of different states: more than 2^{64}. Hence, it will come as no surprise that a model-checker such as JPF will run out of memory when verifying the above very simple code snippet. However, after visiting only 28 states we have already covered an important part of the state space.

Since explicit-state model-checkers generally cannot fully verify the source code of most probabilistic systems, it would be interesting to extend such a model-checker such that it keeps track of the amount of progress it has made with its verification effort. This type of information is much more valuable than a message such as "out of memory."

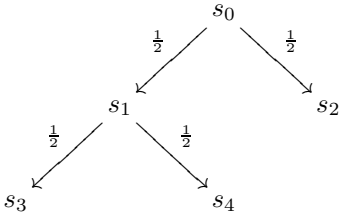
Simply counting the number of (states or) execution paths that have been checked is not very useful for several reasons. First of all, it may be very difficult or even impossible to determine the total number of potential execution paths. Hence, the number of execution paths that have been checked by the model-checker gives us very limited information about the amount of progress that has been made towards verifying the system. Secondly, some execution paths are more likely to happen than others. For example, the nonterminating execution path of the above snippet occurs with probability zero. Checking this execution path amounts to no progress of the verification effort at all.

To capture the progress made by the model-checker, instead of counting the number of execution paths, we endow the set of potential execution paths with a σ-algebra and a probability measure. This is done in a fairly standard way as, for example, in [4]. In this way, we obtain a probability space of execution paths. It assigns probabilities to sets of execution paths. We define our progress measure in terms of the probabilities of certain sets of execution paths. The progress measure returns a number in the interval [0, 1]. This number provides us a quantitative measure of the amount of progress the model-checker has made. The larger the number, the more progress the model-checker has made.

An important property of our progress measure is that it provides a lower bound of the measure of the execution paths that satisfy the property (provided that no counterexample to the property has been found yet). For example, assume that the progress towards verifying the linear-time property ϕ is 0.9999. Then, the probability that we encounter a violation of ϕ when we run the system is at most 0.0001. Hence, even if the model-checker runs out of memory, our progress measure provides useful information about the model-checker's verification effort.

As mentioned earlier, JPF will run out of memory when verifying the Java snippet presented above. However, when checking whether the code can cause overflow, JPF makes 0.9999 progress within a couple of seconds. Note that the code may give rise to overflow, but the probability of that happening is less than one in a googol.

We model the source code of a probabilistic system as a probabilistic transition system. Consider, for example, the following probabilistic transition system.



A model-checker visits the states and transitions of the source code of a probabilistic system represented by a probabilistic transition system in a systematic way. Therefore, we model the verification effort of a model-checker as a sequence of transitions. In this example, we use the indices of the source and target states to name the transitions. For example, the transition from state s_0 to state s_2 is named t_{02}. If the model-checker uses depth-first search to traverse the probabilistic transition system, then it could visit the transitions in, for example, the order t_{01}, t_{13}, t_{14}, t_{02}. If, however, the model-checker uses breadth-first search instead, then it could visit the transitions in, for example, the order t_{01}, t_{02}, t_{13}, t_{14}.

Let us assume that the model-checker verifies the above system using depth-first search. Suppose that the (atomic) proposition p is satisfied in all states. Assume also that we are interested in the linear-time property $\Box p$, that is, we want to check that each state of every execution path satisfies p. In the table below, we present the amount of progress made by the depth-first search.

search	progress
$\emptyset$	0
t_{01}	0
t_{01}, t_{13}	$\frac{1}{4}$
t_{01}, t_{13}, t_{14}	$\frac{1}{2}$
t_{01}, t_{13}, t_{14}, t_{02}	1

Initially, the search is empty, denoted by $\emptyset$. No progress has been made and, hence, the progress is zero. After having traversed transition t_{01}, we still have made no progress since we cannot deduce yet whether any execution path satisfies the property $\Box p$. Therefore, the progress is still zero. Once we have also traversed transition t_{13}, we can conclude that the execution path consisting of the transitions t_{01} and t_{13} satisfies $\Box p$. Since the probability of this execution path is $\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$, the progress increases to $\frac{1}{4}$. After we have traversed transition t_{14}, we can also deduce that the execution path consisting of the transitions

t_{01} and t_{14} satisfies $\Box p$. Because the probability of this execution path is also $\frac{1}{4}$, the progress increases to $\frac{1}{2}$. Finally, we traverse the transition t_{02}. At this point, the whole system has been verified and, hence, the progress is one.

Now suppose that the (atomic) proposition q is satisfied in states s_1 and s_2. This time we want to verify that the linear-time property $\Diamond q$ holds, that is, each execution path contains a state that satisfies q. In the table below, we present the amount of progress made by the model-checker using breadth-first search.

search	progress
$\emptyset$	0
t_{01}	$\frac{1}{2}$
t_{01}, t_{02}	1

After we have traversed transition t_{01}, the progress is $\frac{1}{2}$ since all execution paths that start with t_{01} satisfy the property $\Diamond q$ and the measure of these execution paths is $\frac{1}{2}$. Similarly, once we have also traversed transition t_{02}, the progress reaches one. At this point we can stop our verification effort.

In the table below, we present the amount of progress made by the model-checker towards verifying the linear-time property $\Diamond q$, this time using depth-first search.

search	progress
$\emptyset$	0
t_{01}	$\frac{1}{2}$
t_{01}, t_{13}	$\frac{1}{2}$
t_{01}, t_{13}, t_{14}	$\frac{1}{2}$
t_{01}, t_{13}, t_{14}, t_{02}	1

From the above examples, we can conclude that the linear-time property of interest and the order in which the model-checker traverses the transitions impact the amount of progress made.

When verifying the source code of a (probabilistic) system, we may be interested to check properties such as the absence of uncaught exceptions and the absence of underflow and overflow. All these properties are instances of a special class of linear-time properties called invariants. For the case that the property being verified is such an invariant, we present an alternative characterization of our progress measure. We show that the amount of progress for invariants is the measure of the set of execution paths that have been checked. We present an algorithm that computes the progress for invariants, which is based on this characterization. We have implemented this algorithm within JPF.

2 Probabilistic Transition Systems

We represent the probabilistic system to be verified by the explicit-state model-checker as a probabilistic transition system. The model-checker explores the (states and) transitions of the probabilistic transition system in a systematic way. The set of (states and) transitions of a probabilistic transition system may be countably infinite.

Definition 1. *A probabilistic transition system (PTS for short) is a tuple* $\langle S, T, AP, s_0, \text{source}, \text{target}, \text{prob}, \text{label}\rangle$ *consisting of*

- *a countable set* S *of states,*
- *a countable set* T *of transitions,*
- *a set* AP *of atomic propositions,*
- *an initial state* s_0*,*
- *a function* $\text{source} : T \to S$*,*
- *a function* $\text{target} : T \to S$*,*
- *a function* $\text{prob} : T \to (0, 1]$*, and*
- *a function* $\text{label} : S \to 2^{AP}$

such that

- $s_0 \in S$ *and*
- *for all* $s \in S$*,* $\sum\{\, \text{prob}(t) \mid \text{source}(t) = s \,\} \in \{0, 1\}$.

Instead of $\langle S, T, AP, s_0, \text{source}, \text{target}, \text{prob}, \text{label}\rangle$ we usually write $\mathcal{S}$ and we denote, for example, its set of states by $S_{\mathcal{S}}$. Note that in a PTS, there may be multiple transitions between a pair of states (as can happen in model-checkers such as JPF).

A state is final in $\mathcal{S}$ if it has no outgoing transitions. Next, we formalize the potential execution paths of a PTS. We classify them into two categories: infinite execution paths and finite execution paths.

Definition 2. *An infinite execution path of a PTS* $\mathcal{S}$ *is an infinite sequence of transitions* $t_1 t_2 \ldots$ *such that*

- *for all* $i \geq 1$*,* $t_i \in T_{\mathcal{S}}$*,*
- $\text{source}_{\mathcal{S}}(t_1) = s_{0\mathcal{S}}$*, and*
- *for all* $i \geq 1$*,* $\text{target}_{\mathcal{S}}(t_i) = \text{source}_{\mathcal{S}}(t_{i+1})$.

The set of all infinite execution paths is denoted by $\text{Exec}^{\omega}_{\mathcal{S}}$.

A finite execution path of a PTS $\mathcal{S}$ *is either a finite sequence of transitions* $t_1 \ldots t_n$ *for some* $n \geq 1$ *such that*

- *for all* $1 \leq i \leq n$*,* $t_i \in T_{\mathcal{S}}$*,*
- $\text{source}_{\mathcal{S}}(t_1) = s_{0\mathcal{S}}$*,*
- $\text{target}_{\mathcal{S}}(t_n)$ *is final in* $\mathcal{S}$ *and*
- *for all* $1 \leq i < n$*,* $\text{target}_{\mathcal{S}}(t_i) = \text{source}_{\mathcal{S}}(t_{i+1})$*,*

or the empty sequence ϵ *if* $s_{0\mathcal{S}}$ *is final in* $\mathcal{S}$*. The set of all finite execution paths is denoted by* $\text{Exec}^{*}_{\mathcal{S}}$.

The set of all execution paths $\text{Exec}^{\infty}_{\mathcal{S}}$ *is defined by* $\text{Exec}^{\infty}_{\mathcal{S}} = \text{Exec}^{\omega}_{\mathcal{S}} \cup \text{Exec}^{*}_{\mathcal{S}}$.

As we already discussed in the introduction, we will define our progress measure by means of a measurable space of execution paths. We assume that the reader is familiar with some basic measure theory as can be found in, for example, [2]. Recall that a measurable space consists of a set, in this case the set $\text{Exec}^{\infty}_{\mathcal{S}}$

of executions paths, a σ-algebra and a measure. Our measurable space is very similar[1] to the sequence space as defined, for example, in [4, Chapter 2]. We present only the key ingredients, which will be needed later, and refer the reader to [4] for the details.

In this case, the σ-algebra is a set of subsets of $\mathrm{Exec}_{\mathcal{S}}^{\infty}$. As we will see, this σ-algebra can be generated from the cylinder sets defined next. We denote the set of finite prefixes of execution paths in $\mathrm{Exec}_{\mathcal{S}}^{\infty}$ by $\mathrm{pref}(\mathrm{Exec}_{\mathcal{S}}^{\infty})$.

Definition 3. *Let $e \in \mathrm{pref}(\mathrm{Exec}_{\mathcal{S}}^{\infty})$. Its basic cylinder set $B_{\mathcal{S}}^{e}$ is defined by*

$$B_{\mathcal{S}}^{e} = \{\, e' \in \mathrm{Exec}_{\mathcal{S}}^{\infty} \mid e \text{ is a prefix of } e' \,\}.$$

The set $\mathcal{B}_{\mathcal{S}}$ is defined by $\mathcal{B}_{\mathcal{S}} = \{\, B_{\mathcal{S}}^{e} \mid e \in \mathrm{pref}(\mathrm{Exec}_{\mathcal{S}}^{\infty}) \,\}$.

Note that $B_{\mathcal{S}}^{\epsilon} = \mathrm{Exec}_{\mathcal{S}}^{\infty}$. Next, we recall a measure on $\mathcal{B}_{\mathcal{S}}$.

Definition 4. *The function $\nu_{\mathcal{S}} : \mathcal{B}_{\mathcal{S}} \to [0,1]$ is defined by*

$$\nu_{\mathcal{S}}(B_{\mathcal{S}}^{t_1 \ldots t_n}) = \prod_{1 \le i \le n} \mathrm{prob}_{\mathcal{S}}(t_i).$$

As shown in, for example, [4, Chapter 2], the function $\nu_{\mathcal{S}}$ can be uniquely extended to a probability measure on the σ-algebra generated by $\mathcal{B}_{\mathcal{S}}$. We denote the extended measure by $\mu_{\mathcal{S}}$ and the σ-algebra generated by $\mathcal{B}_{\mathcal{S}}$ by $\Sigma_{\mathcal{S}}$.

We will exploit the probability space $\langle \mathrm{Exec}_{\mathcal{S}}^{\infty}, \Sigma_{\mathcal{S}}, \mu_{\mathcal{S}} \rangle$ to capture our progress measure. In future sections we will use the following properties of this probability space:

- $\Sigma_{\mathcal{S}}$ is a σ-algebra,
- $\mathcal{B}_{\mathcal{S}} \subseteq \Sigma_{\mathcal{S}}$, and
- $\nu_{\mathcal{S}}(B) = \mu_{\mathcal{S}}(B)$ for all $B \in \mathcal{B}_{\mathcal{S}}$.

As we already discussed earlier, we represent the source code of the probabilistic system to be verified by an explicit-state probabilistic model-checker as a PTS. Since the model-checker systematically explores the (states and) transitions of the system under verification, we represent the model-checking as a set of transitions of the PTS.

Definition 5. *A search of a PTS $\mathcal{S}$ is a finite subset of $T_{\mathcal{S}}$.*

Usually, each state $\mathrm{source}_{\mathcal{S}}(t_i)$ of a search $\{t_1, \ldots, t_n\}$ is reachable from the initial state $s_{0\mathcal{S}}$ via the transitions $t_1, \ldots, t_n$. However, this fact is not used in any of our proofs.

[1] The sequence space only consists of infinite sequences whereas we consider both finite and infinite sequences.

3 A Progress Measure

If the model-checker has searched a part of the system, it of course is not aware of the remainder of the system. To capture this, we formalize how a search can be extended.

Definition 6. *The PTS $\mathcal{S}'$ extends the search $\{t_1, \ldots, t_n\}$ of the PTS $\mathcal{S}$ if for all $1 \leq i \leq n$,*

- $t_i \in T_{\mathcal{S}'}$,
- $s_{0\mathcal{S}} = s_{0\mathcal{S}'}$,
- $\text{source}_{\mathcal{S}'}(t_i) = \text{source}_{\mathcal{S}}(t_i)$,
- $\text{target}_{\mathcal{S}'}(t_i) = \text{target}_{\mathcal{S}}(t_i)$,
- $\text{prob}_{\mathcal{S}'}(t_i) = \text{prob}_{\mathcal{S}}(t_i)$,
- $\text{label}_{\mathcal{S}'}(\text{source}_{\mathcal{S}'}(t_i)) = \text{label}_{\mathcal{S}}(\text{source}_{\mathcal{S}}(t_i))$,
- $\text{label}_{\mathcal{S}'}(\text{target}_{\mathcal{S}'}(t_i)) = \text{label}_{\mathcal{S}}(\text{target}_{\mathcal{S}}(t_i))$,
- $\text{target}_{\mathcal{S}'}(t_i)$ *is final in* $\mathcal{S}'$ *iff* $\text{target}_{\mathcal{S}}(t_i)$ *is final in* $\mathcal{S}$, *and*
- $s_{0_{\mathcal{S}}}$ *is final in* $\mathcal{S}'$ *iff* $s_{0_{\mathcal{S}'}}$ *is final in* $\mathcal{S}$.

To avoid clutter, instead of $s_{0\mathcal{S}}$, we will write s_0 in the remainder of this paper. Note that the PTS $\mathcal{S}$ itself extends the search $\{t_1, \ldots, t_n\}$ of $\mathcal{S}$.

In Definition 9, we will define the amount of progress a search has made towards verifying a linear-time property. This amount only makes sense as long as no violation of the property has been found. The latter is formalized next.

Definition 7. *The search $\{t_1, \ldots, t_n\}$ of the PTS $\mathcal{S}$ has not found a violation of the linear-time property ϕ if $e \models_{\mathcal{S}} \phi$ for all $e \in \text{Exec}^{\infty}_{\mathcal{S}} \cap \{t_1, \ldots, t_n\}^{\infty}$.*

The above introduced notion is natural and rather weak. It suffices for all our results apart from Theorem 2. There we use a slightly stronger notion.

Definition 8. *Let the PTS $\mathcal{S}'$ extend the search $\{t_1, \ldots, t_n\}$ of PTS $\mathcal{S}$ and let ϕ be a linear-time property. The set $\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n)$ is defined by*

$$\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n) = \bigcup \{\, B^e_{\mathcal{S}'} \mid e \in \{t_1, \ldots, t_n\}^* \wedge \forall e' \in B^e_{\mathcal{S}'} : e' \models_{\mathcal{S}'} \phi \,\}.$$

The set $\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n)$ is the union of basic cylinder sets $B^e_{\mathcal{S}'}$ whose execution paths, that is, all extensions of e in $\mathcal{S}'$, satisfy ϕ. In that case, $B^e_{\mathcal{S}'}$ does not contain a counterexample of ϕ. The set $\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n)$ is measurable, as is stated in the following proposition.

Proposition 1. *Let ϕ be a linear-time property, and the PTS $\mathcal{S}'$ extend the search $\{t_1, \ldots, t_n\}$ of the PTS $\mathcal{S}$. Then $\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n) \in \Sigma_{\mathcal{S}'}$.*

Since the set $\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n)$ is measurable, its measure $\mu_{\mathcal{S}'}(\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n))$ is defined. The latter is a number in the interval $[0, 1]$ which represents the "size" of the basic cylinder sets that do not contain a counterexample of ϕ. This number captures the amount of progress of $\{t_1, \ldots, t_n\}$ for ϕ, *provided that* the PTS under consideration is $\mathcal{S}'$. Since we have no knowledge of the transitions other than the search, we consider that all extensions $\mathcal{S}'$ of $\{t_1, \ldots, t_n\}$ and consider the worst case in terms of progress.

Definition 9. *Assume that the search* $\{t_1, \ldots, t_n\}$ *of the PTS* $\mathcal{S}$ *has not found a violation of the linear-time property* ϕ*. The progress of* $\{t_1, \ldots, t_n\}$ *for* ϕ *is defined by*

$$\mathrm{prog}_{\mathcal{S}}(t_1, \ldots, t_n, \phi) = \inf \left\{ \mu_{\mathcal{S}'} \left(\mathcal{B}^{\phi}_{\mathcal{S}'}(t_1, \ldots, t_n) \right) \mid \mathcal{S}' \textit{ extends } \{t_1, \ldots, t_n\} \textit{ of } \mathcal{S} \right\}.$$

Next, we prove a key property of our progress measure. We show that it is a lower bound for the probability that the linear-time property holds in $\mathcal{S}$. For example, if the progress measure is 0.9999, we know that the probability that the property holds is at least 0.9999. As a consequence, the probability that we encounter a violation of the property when we run the system is at most 0.0001.

Theorem 1. *Assume that the search* $\{t_1, \ldots, t_n\}$ *of the PTS* $\mathcal{S}$ *has not found a violation of the linear-time property* ϕ*. Then*

$$\mathrm{prog}_{\mathcal{S}}(t_1, \ldots, t_n, \phi) \leq \mu_{\mathcal{S}}(\{\, e \in \mathrm{Exec}^{\infty}_{\mathcal{S}} \mid e \models_{\mathcal{S}} \phi \,\}).$$

Note that the above result only holds for linear-time properties ϕ that are measurable for $\mathcal{S}$, that is, $\{\, e \in \mathrm{Exec}^{\infty}_{\mathcal{S}} \mid e \models_{\mathcal{S}} \phi \,\} \in \Sigma_{\mathcal{S}}$. Vardi [8, Corollary 2.4] has shown that all properties expressed in LTL are measurable.

4 Characterization of Progress for Invariants

In this section, we restrict our attention to the case that the linear-time property is an invariant. In particular, we assume that the property is of the form $\Box p$ for some atomic proposition p. Invariants form an important class of linear-time properties. In particular for source code, this class plays a key role. For example, we may want to check that the code never gives rise to any uncaught exceptions, or that it never causes overflow. These types of properties can all be expressed as invariants.

As we already mentioned, in our progress measure we consider the worst case. Next, we show that the best case, obtained by replacing the inf in the definition of our progress measure with a sup, does not provide any useful information. We consider a slightly stronger, yet still realistic, notion of a search not having a found a violation of the invariant $\Box p$: all states explored by the search satisfy the atomic proposition p. To avoid clutter, we use $S^{t_1,\ldots,t_n}_{\mathcal{S}}$ to denote the set

$$\{\, \mathrm{source}_{\mathcal{S}}(t_i) \mid 1 \leq i \leq n \,\} \cup \{\, \mathrm{target}_{\mathcal{S}}(t_i) \mid 1 \leq i \leq n \,\} \cup \{s_0\}.$$

Theorem 2. *Consider the search* $\{t_1, \ldots, t_n\}$ *of the PTS* $\mathcal{S}$*. Assume that* $p \in \mathrm{label}_{\mathcal{S}}(s)$ *for all* $s \in S^{t_1,\ldots,t_n}_{\mathcal{S}}$*. Then*

$$\sup \left\{ \mu_{\mathcal{S}'} \left(\mathcal{B}^{\Box p}_{\mathcal{S}'}(t_1, \ldots, t_n) \right) \mid \mathcal{S}' \textit{ extends } \{t_1, \ldots, t_n\} \textit{ of } \mathcal{S} \right\} = 1.$$

From the above theorem we can derive that the best upper bound for $\mu_{\mathcal{S}}(\{\, e \in \mathrm{Exec}^{\infty}_{\mathcal{S}} \mid e \models_{\mathcal{S}} \phi \,\})$ is one in that case.

In the remainder of this section, we provide a characterization of our progress measure for invariants. This characterization is the basis for the algorithm to compute the progress measure for invariants presented in Section 5. Given a search $\{t_1, \ldots, t_n\}$ of the PTS $\mathcal{S}$, we show that the progress of $\{t_1, \ldots, t_n\}$ for an invariant for which no violation has been found yet is the measure of the set $\mathrm{Exec}^{\infty}_{\mathcal{S}} \cap \{t_1, \ldots, t_n\}^{\infty}$, where $\{t_1, \ldots, t_n\}^{\infty}$ denotes the set of finite and infinite sequences of the transition $t_1, \ldots, t_n$. This set is measurable as shown in the following proposition.

Proposition 2. *Let the PTS $\mathcal{S}'$ extend the search $\{t_1, \ldots, t_n\}$ of the PTS $\mathcal{S}$. Then* $\mathrm{Exec}^{\infty}_{\mathcal{S}'} \cap \{t_1, \ldots, t_n\}^{\infty} \in \Sigma_{\mathcal{S}'}$.

We prove the characterization by showing two inequalities. In the first proof, we construct a PTS $\mathcal{S}'$ that extends search $\{t_1, \ldots, t_n\}$ of the PTS $\mathcal{S}$ such that

$$\mu_{\mathcal{S}'}\left(\mathcal{B}^{\Box p}_{\mathcal{S}'}(t_1, \ldots, t_n)\right) \leq \mu_{\mathcal{S}}(\mathrm{Exec}^{\infty}_{\mathcal{S}} \cap \{t_1, \ldots, t_n\}^{\infty}).$$

We construct the system $\mathcal{S}'$ by

- adding an extra state $s_\perp$ with $p \notin \mathrm{label}_{\mathcal{S}'}(s_\perp)$,
- adding a transition $t_\perp$ with $\mathrm{source}_{\mathcal{S}'}(t_\perp) = s_\perp$, $\mathrm{target}_{\mathcal{S}'}(t_\perp) = s_\perp$ and $\mathrm{prob}_{\mathcal{S}'}(t_\perp) = 1$, and
- adding an extra transition t_s with $\mathrm{source}_{\mathcal{S}'}(t_s) = s$, $\mathrm{target}_{\mathcal{S}'}(t_s) = s_\perp$, and $\mathrm{prob}_{\mathcal{S}'}(t_s) = 1 - \mathrm{out}_{\mathcal{S}}(s)$ for all states $s \in S^{t_1,\ldots,t_n}_{\mathcal{S}}$ such that $\mathrm{out}_{\mathcal{S}}(s) < 1$ and s is not final in $S_{\mathcal{S}'}$, where

$$\mathrm{out}_{\mathcal{S}}(s) = \sum \{\, \mathrm{prob}_{\mathcal{S}}(t_i) \mid 1 \leq i \leq n \wedge \mathrm{source}_{\mathcal{S}}(t_i) = s \,\}.$$

Lemma 1. *Assume that the search $\{t_1, \ldots, t_n\}$ of the PTS $\mathcal{S}$ has not found a violation of the invariant $\Box p$. Then*

$$\mathrm{prog}_{\mathcal{S}}(t_1, \ldots, t_n, \Box p) \leq \mu_{\mathcal{S}}(\mathrm{Exec}^{\infty}_{\mathcal{S}} \cap \{t_1, \ldots, t_n\}^{\infty}).$$

An obvious attempt to prove the other inequality by showing

$$\mathrm{Exec}^{\infty}_{\mathcal{S}} \cap \{t_1, \ldots, t_n\}^{\infty} \subseteq \mathcal{B}^{\Box p}_{\mathcal{S}'}(t_1, \ldots, t_n)$$

for each PTS $\mathcal{S}'$ that extends the search $\{t_1, \ldots, t_n\}$ of the PTS $\mathcal{S}$ fails, because the set

$$(\mathrm{Exec}^{\infty}_{\mathcal{S}} \cap \{t_1, \ldots, t_n\}^{\omega}) \setminus \mathcal{B}^{\Box p}_{\mathcal{S}'}(t_1, \ldots, t_n).$$

is not always empty. However, as we will show below, the above set has measure zero.

Lemma 2. *Let the PTS $\mathcal{S}'$ extend the search $\{t_1, \ldots, t_n\}$ of the PTS $\mathcal{S}$. Then*

$$\mu_{\mathcal{S}'}\left((\mathrm{Exec}^{\infty}_{\mathcal{S}'} \cap \{t_1, \ldots, t_n\}^{\omega}) \setminus \mathcal{B}^{\Box p}_{\mathcal{S}'}(t_1, \ldots, t_n)\right) = 0.$$

By means of the above lemma we can now prove the other inequality.

Lemma 3. *Assume that the search $\{t_1, \ldots, t_n\}$ of the PTS $\mathcal{S}$ has not found a violation of the invariant $\Box p$. Then*

$$\mu_{\mathcal{S}}(\mathrm{Exec}^{\infty}_{\mathcal{S}} \cap \{t_1, \ldots, t_n\}^{\infty}) \leq \mathrm{prog}_{\mathcal{S}}(t_1, \ldots, t_n, \Box p).$$

Combining the above results, we arrive at the characterization of our progress measure for invariants.

Theorem 3. *Assume that the search $\{t_1, \ldots, t_n\}$ of the PTS $\mathcal{S}$ has not found a violation of the invariant $\Box p$. Then*

$$\mathrm{prog}_{\mathcal{S}}(t_1, \ldots, t_n, \Box p) = \mu_{\mathcal{S}}(\mathrm{Exec}^{\infty}_{\mathcal{S}} \cap \{t_1, \ldots, t_n\}^{\infty}).$$

Note that if the search $\{t_1, \ldots, t_n\}$ of the PTS $\mathcal{S}$ has not found a violation of the invariant $\Box p$, then $\mathrm{prog}_{\mathcal{S}}(t_1, \ldots, t_n, \Box p)$ is independent of the atomic proposition p.

5 Computation of Progress for Invariants

Given the characterization of our progress measure for invariants, we are now in a position to compute the progress measure for invariants. From a search $\{t_1, \ldots, t_n\}$ of a PTS $\mathcal{S}$ we construct a PTS $\mathcal{S}'$. From $\mathcal{S}$ we remove all transitions different from $t_1, \ldots, t_n$ and we add a transition t_s from state s to a new state $s_\perp$ for all states s which have not been fully explored yet (that is, in $\mathcal{S}$ such a state s has outgoing transitions different from $t_1, \ldots, t_n$).

Definition 10. *Let $\{t_1, \ldots, t_n\}$ be a search of the PTS $\mathcal{S}$ and let $p \in AP_{\mathcal{S}}$. Then the PTS $\mathcal{S}'$ is defined as follows. The set $S_{\mathcal{S}'}$ is defined by $S_{\mathcal{S}'} = S_{\mathcal{S}}^{t_1,\ldots,t_n} \cup \{s_\perp\}$. The set $T_{\mathcal{S}'}$ is defined by*

$$T_{\mathcal{S}'} = \{t_1, \ldots, t_n\} \cup \{\, t_s \mid s \in S_{\mathcal{S}'} \text{ is not final in } \mathcal{S} \text{ and } \mathrm{out}_{\mathcal{S}}(s) < 1 \,\} \cup \{t_\perp\}.$$

The set $AP_{\mathcal{S}'}$ is defined by $AP_{\mathcal{S}'} = AP_{\mathcal{S}}$. The function $\mathrm{source}_{\mathcal{S}'} : T_{\mathcal{S}'} \to S_{\mathcal{S}'}$ is defined by

$$\mathrm{source}_{\mathcal{S}'}(t) = \begin{cases} s_\perp & \text{if } t = t_\perp \\ s & \text{if } t = t_s \\ \mathrm{source}_{\mathcal{S}}(t) & \text{if } t \in \{t_1, \ldots, t_n\}. \end{cases}$$

The function $\mathrm{target}_{\mathcal{S}'} : T_{\mathcal{S}'} \to S_{\mathcal{S}'}$ is defined by

$$\mathrm{target}_{\mathcal{S}'}(t) = \begin{cases} s_\perp & \text{if } t = t_\perp \\ s_\perp & \text{if } t = t_s \\ \mathrm{target}_{\mathcal{S}}(t) & \text{if } t \in \{t_1, \ldots t_n\}. \end{cases}$$

The function $\mathrm{prob}_{\mathcal{S}'} : T_{\mathcal{S}'} \to (0, 1]$ is defined by

$$\mathrm{prob}_{\mathcal{S}'}(t) = \begin{cases} 1 & \text{if } t = t_\perp \\ 1 - \mathrm{out}_{\mathcal{S}}(s) & \text{if } t = t_s \\ \mathrm{prob}_{\mathcal{S}}(t) & \text{if } t \in \{t_1, \ldots t_n\}. \end{cases}$$

The function $\text{label}_{\mathcal{S}'} : S_{\mathcal{S}'} \to 2^{AP_{\mathcal{S}'}}$ *is defined by*

$$\text{label}_{\mathcal{S}'}(s) = \begin{cases} \emptyset & \text{if } s = s_\perp \\ \text{label}_{\mathcal{S}}(s) & \text{otherwise.} \end{cases}$$

As we show next, to compute the progress of the search $\{t_1, \ldots, t_n\}$ of the PTS $\mathcal{S}$ towards verifying the invariant $\Box p$, it suffices to compute the measure of the set of those execution paths of the PTS $\mathcal{S}'$ that contain the transition $t_\perp$.

Theorem 4. *Assume that the search* $\{t_1, \ldots, t_n\}$ *of the PTS* $\mathcal{S}$ *has not found a violation of the invariant* $\Box p$*. Consider the PTS* $\mathcal{S}'$ *introduced in Definition 10. Then*

$$\text{prog}_{\mathcal{S}}(t_1, \ldots, t_n, \Box p) = 1 - \mu_{\mathcal{S}'}(\{\, e \in \text{Exec}^{\infty}_{\mathcal{S}'} \mid e \text{ contains } t_\perp \}).$$

Obviously, an execution path of $\mathcal{S}'$ contains the transition $t_\perp$ iff it reaches the state $s_\perp$. Hence, it suffices to compute the measure of the set of those execution paths of $\mathcal{S}'$ that reach the state $s_\perp$. Several algorithms and tools are available to compute the probability of reaching a particular state (see, for example, [1, Section 10.1]).

6 Conclusion

To measure the amount of progress an explicit-state probabilistic model-checker has made towards verifying a linear-time property, we introduced the notion of a progress measure. Although our notion is based on a probability space that has been used by others to study probabilistic systems, as far as we know this notion is new. We believe that the simplicity of our approach has some advantages: the theory may be easily extensible (some directions for further research are discussed below) and it may give rise to efficient algorithms. Another contribution of this paper is an algorithm to compute the progress measure for invariants. We have implemented our theoretical framework within JPF. The implementation details and some experimental results can be found in [10].

The work most closely related to ours is the work by Pavese, Braberman and Uchitel [6].[2] In their position paper, they sketch how to provide useful feedback when a model-checker runs out of memory. They aim to measure the probability that a run of the system reaches a state that has not be visited by the model-checker. They do not take the property being verified into account and limit the depth of the search. In [7], Della Penna et al. show how, given a Markov chain and an integer i, the probability of reaching a state s within i transitions can be computed. We conjecture that this is closely related to computing the progress of breadth-first search verifying the linear-time property $\Diamond p$, where the atomic proposition p is only satisfied in state s.

[2] Our work was carried out independently from theirs. A preliminary version of our work appeared in [11]. We only became aware of a draft version of [6] while finishing [10].

Our framework only considers probabilistic choices. As a consequence, it is only applicable to sequential randomized algorithms. One direction of future work is to extend our framework so that it can handle concurrent randomized algorithms as well. In that case, we also have to deal with nondeterministic choices and, hence, we have to consider schedulers.

Although invariants form an important class of linear-time properties, we are interested in developing algorithms to compute the progress measure for other classes of linear-time properties as well. In this paper, we focused on explicit-state probabilistic model-checkers. However, we believe that the theory we developed can be adapted to symbolic probabilistic model-checkers. This is another direction for future research.

Acknowledgements. We thank all the referees of this paper for their constructive feedback.[3] We are also thankful to the members of the DisCoVeri group for discussion.

References

1. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, Cambridge (2008)
2. Billingsley, P.: Probability and Measure. John Wiley & Sons, Chichester (1995)
3. Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Performance Evaluation 68(2), 90–104 (2011)
4. Kemeny, J.G., Snell, J.L., Knapp, A.W.: Denumerable Markov Chains. Van Nostrand, New York (1966)
5. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic symbolic model checking with PRISM: A hybrid approach. International Journal on Software Tools for Technology Transfer 6(2), 128–142 (2004)
6. Pavese, E., Braberman, V., Uchitel, S.: My model checker died!: how well did it do? In: Proceedings of the 2010 ICSE Workshop on Quantitative Stochastic Models in the Verification and Design of Software Systems, pp. 33–40. ACM, New York (2010)
7. Della Penna, G., Intrigila, B., Melatti, I., Tronci, E., Venturini Zilli, M.: Finite horizon analysis of Markov chains with the Murφ verifier. International Journal on Software Tools for Technology 8(4/5), 397–409 (2006)
8. Vardi, M.Y.: Automatic verification of probabilistic finite-state programs. In: Proceedings of the 26th IEEE Symposium on Foundations of Computer Science, pp. 327–338. IEEE, Los Alamitos (1985)
9. Visser, W., Havelund, K., Brat, G., Park, S., Lerda, F.: Model checking programs. Automated Software Engineering 10(2), 203–232 (2003)
10. Zhang, X.: Measuring Progress of Model Checking Randomized Algorithms. Master's thesis. York University, Toronto (2010)
11. Zhang, X., Breugel, F.v.: Measuring progress of Java PathFinder model-checking randomized sequential code. In: Preliminary Proceedings of the 6th Workshop on Quantitative Aspects of Programming Languages, 4 pages (2008)

[3] Due to lack of space, some suggestions could unfortunately not be implemented.

Probabilistic Bisimulation and Simulation Algorithms by Abstract Interpretation

Silvia Crafa and Francesco Ranzato

University of Padova, Italy

Abstract. We show how bisimulation equivalence and simulation preorder on probabilistic LTSs (PLTSs), namely the main behavioural relations on probabilistic nondeterministic processes, can be characterized by abstract interpretation. Both bisimulation and simulation can be obtained as completions of partitions and preorders, viewed as abstract domains, w.r.t. a pair of concrete functions that encode a PLTS. As a consequence, this approach provides a general framework for designing algorithms for computing bisimulation and simulation on PLTSs. Notably, (i) we show that the standard bisimulation algorithm by Baier et al. can be viewed as an instance of such a framework and (ii) we design a new efficient simulation algorithm that improves the state of the art.

1 Introduction

Randomization phenomena in concurrent systems have been widely studied in probabilistic extensions of process algebras like Markov chains and probabilistic labeled transition systems (PLTSs). Most standard tools for studying nonprobabilistic processes, like behavioural equivalences, temporal logics and model checking, have been investigated for these probabilistic models. In particular, bisimulation equivalence and simulation preorder relations, namely the main behavioural relations between concurrent systems, have been extended and studied in a probabilistic setting [5,7,12].

Abstract interpretation [2,3] is a well-known general theory for specifying the approximation of formal semantics. Abstract domains play an essential role in any abstract interpretation design, since they encode in an ordered structure how concrete semantic properties are approximated. A number of behavioural relations, including bisimulation, stuttering bisimulation and simulation, have been characterized in abstract interpretation as complete refinements, called forward complete shells, of abstract domains w.r.t. logical/temporal operators of suitable modal logics [11]. One notable benefit of this approach is that it provides a general framework for designing basic algorithms that compute behavioral relations as forward complete shells of abstract domains. As a remarkable example, this abstract interpretation-based approach led to an efficient algorithm for computing the simulation preorder [10] that features the best time complexity among the simulation algorithms.

This paper extends the aforementioned results to a probabilistic setting. In particular, we consider probabilistic processes specified as PLTSs, a general model that exhibits both non-deterministic choice (as in LTSs) and probabilistic choice (as in Markov chains). In [11], bisimulation in LTSs has been characterized in terms of forward complete shells of partitions w.r.t. the predecessor operator of LTSs. We show that this same

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 295–306, 2011.

idea scales to the case of PLTSs by considering the probabilistic predecessor operator that defines the transitions of a PLTS together with a probabilistic function that encodes the distributions in the PLTS (this latter operator is somehow reminiscent of a probabilistic connective in Parma and Segala's [9] modal logics for probabilistic bisimulation and simulation). Bisimulation equivalence in PLTSs is then characterized as a domain refinement through a complete shell w.r.t. the above two operators. On the other hand, the simulation preorder in PLTSs turns out to be the same complete shell of abstract domains w.r.t. the same two operators, but using different underlying abstract domains: for bisimulation, the complete shell is computed in a space of abstractions that are state and distribution partitions, while for simulation the same complete shell is instead computed over abstractions that are preorders on states and distributions.

Complete shells of abstract domains may in general be obtained through a simple fixpoint computation. We show how such a basic procedure can be instantiated to obtain two algorithms that iteratively compute bisimulation and simulation on PLTSs. Interestingly, the standard procedure for computing bisimulations in PLTSs, namely Baier-Engelen-Majster's algorithm [1], can be actually viewed as an implementation of our complete shell procedure that characterizes bisimulation. On the other hand, we show that the corresponding complete shell for computing the simulation preorder yields a new efficient probabilistic simulation algorithm that advances the state-of-the-art: in fact, its time and space complexity bounds improve on the best known simulation algorithm for PLTSs by Zhang et al. [13].

2 Bisimulation and Simulation in PLTSs

Given a set X, $\mathrm{Distr}(X)$ denotes the set of (stochastic) distributions on X, i.e., functions $d{:}X \to [0,1]$ such that $\sum_{x\in X} d(x) = 1$. The support of a distribution d is defined by $\mathrm{supp}(d) \triangleq \{x \in X \mid d(x) > 0\}$; also, if $S \subseteq X$, then $d(S) \triangleq \sum_{s\in S} d(s)$.

A probabilistic LTS (PLTS) is a tuple $\mathcal{S} = \langle \Sigma, Act, \to\rangle$ where Σ is a set of states, Act is a set of actions and $\to \subseteq \Sigma \times Act \times \mathrm{Distr}(\Sigma)$ is a transition relation, where $(s,a,d) \in \to$ is also denoted by $s \xrightarrow{a} d$. We denote by $\mathrm{Distr} \triangleq \{d \in \mathrm{Distr}(\Sigma) \mid \exists s \in \Sigma. \exists a \in Act\,.\, s \xrightarrow{a} d\}$ the set of target distributions in $\mathcal{S}$. Given $D \subseteq \mathrm{Distr}$, we write $s \xrightarrow{a} D$ when there exists $d \in D$ such that $s \xrightarrow{a} d$. For any $a \in Act$, the predecessor and successor operators $\mathrm{pre}_a : \wp(\mathrm{Distr}) \to \wp(\Sigma)$ and $\mathrm{post}_a : \wp(\Sigma) \to \wp(\mathrm{Distr})$ are defined by $\mathrm{pre}_a(D) \triangleq \{s \in \Sigma \mid s \xrightarrow{a} D\}$ and $\mathrm{post}_a(S) \triangleq \{d \in \mathrm{Distr} \mid \exists s \in S.\, s \xrightarrow{a} d\}$. For any $d \in \mathrm{Distr}$ and $s \in \Sigma$, we define $\mathrm{in}(d) \triangleq \{a \in Act \mid \mathrm{pre}_a(d) \neq \varnothing\}$ and $\mathrm{out}(s) \triangleq \{a \in Act \mid \mathrm{post}_a(s) \neq \varnothing\}$.

Bisimulation. Let $\mathrm{Part}(X)$ denote the set of partitions of a finite set X. If $P \in \mathrm{Part}(X)$ and $x \in X$ then $P(x)$ denotes the unique block of P that contains x. A partition P induces a mapping $P : \wp(X) \to \wp(X)$ defined as $P(Y) \triangleq \cup_{y\in Y} P(y)$. Any partition $P \in \mathrm{Part}(X)$ induces an equivalence relation (i.e., a partition) over distributions $\equiv_P \in \mathrm{Part}(\mathrm{Distr}(X))$ as follows: for any $d, e \in \mathrm{Distr}(X)$, $d \equiv_P e$ if for any $B \in P$, $d(B) = e(B)$. In words, two distributions are $\equiv_P$-equivalent whenever they give the same probability to the blocks of P.

Given a PLTS $\mathcal{S} = \langle \Sigma, Act, \to\rangle$, a partition $P \in \mathrm{Part}(\Sigma)$ is a bisimulation on $\mathcal{S}$ when for all $s, t \in \Sigma$ and $d \in \mathrm{Distr}$, if $P(s) = P(t)$ and $s \xrightarrow{a} d$ then there exists $e \in \mathrm{Distr}$

such that $t \xrightarrow{a} e$ and $d \equiv_P e$. Bisimilarity $P_{\mathrm{bis}} \in \mathrm{Part}(\Sigma)$ is defined as: $P_{\mathrm{bis}}(s) \triangleq \cup \{P(s) \mid P \text{ is a bisimulation on } \mathbb{S}\}$. P_{bis} turns out to be the greatest bisimulation on $\mathbb{S}$ which is also called the bisimulation partition on $\mathbb{S}$.

Simulation. Let $\mathrm{PreOrd}(X)$ denote the set of preorders on X. If $R \in \mathrm{PreOrd}(X)$ and $S \subseteq X$ then $R(S) \triangleq \{x \in X \mid \exists s \in S.\, (s,x) \in R\}$. Similarly to the case of partitions, any preorder $R \in \mathrm{PreOrd}(X)$ induces a preorder $\leq_R$ on $\mathrm{Distr}(X)$ as follows: for any $d, e \in \mathrm{Distr}(X)$, $d \leq_R e$ if for any $S \subseteq X$, $d(S) \leq e(R(S))$. Such a definition of $\leq_R$ can be equivalently stated in terms of so-called weight functions between distributions and of maximum flows between networks. In particular, it turns out that $d \leq_R e$ iff the maximum flow of a suitable bipartite network built over the states in $\mathrm{supp}(d) \cup \mathrm{supp}(e)$ and over the relation R is 1 (see [1,13]).

A preorder $R \in \mathrm{PreOrd}(\Sigma)$ is a simulation on a PLTS $\mathbb{S}$ when for all $s, t \in \Sigma$ and $d \in \mathrm{Distr}$, if $t \in R(s)$ and $s \xrightarrow{a} d$ then there exists $e \in \mathrm{Distr}$ such that $t \xrightarrow{a} e$ and $d \leq_R e$. The simulation preorder $R_{\mathrm{sim}} \in \mathrm{PreOrd}(\Sigma)$ on $\mathbb{S}$ is defined as follows: $R_{\mathrm{sim}}(s) \triangleq \cup\{R(s) \mid R \text{ is a simulation on } \mathbb{S}\}$. It turns out that R_{sim} is the greatest simulation preorder on $\mathbb{S}$. Simulation partition P_{psim} on $\mathbb{S}$ is the kernel of the simulation preorder, i.e., $P_{\mathrm{psim}}(s) = P_{\mathrm{psim}}(t)$ iff $s \in R_{\mathrm{sim}}(t)$ and $t \in R_{\mathrm{sim}}(s)$.

3 Shells

Forward Completeness. In standard abstract interpretation [2,3], approximations of a concrete semantic domain are encoded by abstract domains (or abstractions), that are specified by Galois insertions (GIs for short) or, equivalently, by adjunctions. A GI of an abstract domain $\langle A, \leq_A \rangle$ into a concrete domain $\langle C, \leq_C \rangle$ is determined by a surjective abstraction map $\alpha : C \to A$ and a 1-1 concretization map $\gamma : A \to C$ such that $\alpha(c) \leq_A a \Leftrightarrow c \leq_C \gamma(a)$, and is denoted by (α, C, A, γ). Recall that GIs of a common concrete domain C are preordered w.r.t. their relative precision: $\mathcal{G}_1 = (\alpha_1, C, A_1, \gamma_1) \trianglelefteq \mathcal{G}_2 = (\alpha_2, C, A_2, \gamma_2)$ — i.e. A_1/A_2 is a refinement/simplification of A_2/A_1 — iff $\gamma_1 \circ \alpha_1 \sqsubseteq_{C \to C} \gamma_2 \circ \alpha_2$. Moreover, $\mathcal{G}_1$ and $\mathcal{G}_2$ are equivalent when $\mathcal{G}_1 \trianglelefteq \mathcal{G}_2$ and $\mathcal{G}_2 \trianglelefteq \mathcal{G}_1$. We denote by $\mathrm{Abs}(C)$ the family of abstract domains of C up to the above equivalence. It is well known that $\langle \mathrm{Abs}(C), \trianglelefteq \rangle$ is a complete lattice. Given a family of abstract domains $\mathcal{X} \subseteq \mathrm{Abs}(C)$, their lub $\sqcup\mathcal{X}$ is the most precise domain in $\mathrm{Abs}(C)$ which is a simplification of any domain in $\mathcal{X}$.

Let $f : C \to D$ be some concrete semantic function, and let $A \in \mathrm{Abs}(C)$ and $B \in \mathrm{Abs}(D)$ be abstractions of the concrete domains C and D. Given an abstract function $f^\sharp : A \to B$, we have that $\langle A, B, f^\sharp \rangle$ is a sound abstract interpretation of f when $f \circ \gamma_{A,C} \sqsubseteq_{A \to D} \gamma_{B,D} \circ f^\sharp$. Forward completeness [3,4] corresponds to the following strengthening of soundness: $f \circ \gamma_{A,C} = \gamma_{B,D} \circ f^\sharp$, meaning that the abstract function $f^\sharp$ is able to replicate the behaviour of f on the abstract domains A and B with no loss of precision. It turns out that if $\langle A, B, f^\sharp \rangle$ is forward complete then the abstract function $f^\sharp$ indeed coincides with $\alpha_{D,B} \circ f \circ \gamma_{A,C}$, that is the best correct approximation of the concrete function f on the pair of abstractions $\langle A, B \rangle$. Hence, the notion of forward completeness of an abstract interpretation does not depend on the choice of the abstract function $f^\sharp$ but only depends on the abstract domains A and B. Accordingly, a pair of abstract domains $\langle A, B \rangle \in \mathrm{Abs}(C) \times \mathrm{Abs}(D)$ is called forward complete for f

```
ShellAlgo(F, G, A, B) {
  Initialize();
  while ¬(F-Stable ∧ G-Stable) do
    if ¬F-Stable then G-Stable := Stabilize(F, A, B); F-Stable := true;
    if ¬G-Stable then F-Stable := Stabilize(G, B, A); G-Stable := true;
}

Initialize() {
  F-Stable := CheckStability(F, A, B); G-Stable := CheckStability(G, B, A);
}

bool Stabilize(H, X, Y) {
  Y_old := Y;
  Y := ⊔{Y' ∈ Abs | Y' ⊴ Y, ⟨X, Y'⟩ is H-complete};
  return (Y = Y_old);
}
```

Fig. 1. Basic Shell Algorithm

(or simply f-complete) iff $f \circ \gamma_{A,C} = \gamma_{B,D} \circ (\alpha_{D,B} \circ f \circ \gamma_{A,C})$. Equivalently, $\langle A, B\rangle$ is f-complete iff the image of f in D, that is $f(\gamma_{A,C}(A))$, is contained in $\gamma_{B,D}(B)$. If $\mathcal{F} \subseteq C \to D$ is a set of concrete functions then $\langle A, B\rangle$ is $\mathcal{F}$-complete when $\langle A, B\rangle$ is f-complete for all $f \in \mathcal{F}$.

Shells of Abstract Domains. Given a set of semantic functions $\mathcal{F} \subseteq C \to D$ and a pair of abstractions $\langle A, B\rangle \in \mathrm{Abs}(C) \times \mathrm{Abs}(D)$, the notion of forward complete shell [4] formalizes the problem of finding the most abstract pair $\langle A', B'\rangle$ such that $A' \trianglelefteq A$, $B' \trianglelefteq B$ and $\langle A', B'\rangle$ is $\mathcal{F}$-complete, which is a particular case of abstraction refinement. It turns out (see [4]) that any pair $\langle A, B\rangle$ can be minimally refined to its forward $\mathcal{F}$-complete shell $\mathrm{Shell}_{\mathcal{F}}(\langle A, B\rangle) \triangleq \sqcup\{\langle A', B'\rangle \in \mathrm{Abs}(C) \times \mathrm{Abs}(D) \mid \langle A', B'\rangle \trianglelefteq \langle A, B\rangle,\ \langle A', B'\rangle \text{ is } \mathcal{F}\text{-complete}\}$. Thus, $\mathrm{Shell}_{\mathcal{F}}(\langle A, B\rangle)$ encodes the least refinement of a pair of abstractions $\langle A, B\rangle$ which is needed in order to obtain forward completeness for $\mathcal{F}$.

Let us now consider a further set of concrete semantic functions $\mathcal{G} \subseteq D \to C$ that operate in the opposite direction w.r.t. $\mathcal{F}$, i.e., from D to C. Given $A \in \mathrm{Abs}(C)$ and $B \in \mathrm{Abs}(D)$, it makes sense to consider both forward $\mathcal{F}$-completeness of $\langle A, B\rangle$ and forward $\mathcal{G}$-completeness of the reversed pair $\langle B, A\rangle$. Thus, $\langle A, B\rangle$ is defined to be $\langle \mathcal{F}, \mathcal{G}\rangle$-complete when $\langle A, B\rangle$ is $\mathcal{F}$-complete and $\langle B, A\rangle$ is $\mathcal{G}$-complete. Here again, any pair $\langle A, B\rangle$ can be minimally refined to its $\langle \mathcal{F}, \mathcal{G}\rangle$-complete shell $\mathrm{Shell}_{\langle \mathcal{F},\mathcal{G}\rangle}(\langle A, B\rangle) \triangleq \sqcup\{\langle A', B'\rangle \in \mathrm{Abs}(C) \times \mathrm{Abs}(D) \mid \langle A', B'\rangle \trianglelefteq \langle A, B\rangle,\ \langle A', B'\rangle \text{ is } \langle \mathcal{F}, \mathcal{G}\rangle\text{-complete}\}$.

The combined shell $\mathrm{Shell}_{\langle \mathcal{F},\mathcal{G}\rangle}(\langle A, B\rangle)$ can be obtained through the ShellAlgo() procedure described in Figure 1. This procedure ShellAlgo() crucially relies on the Stabilize() function: given a set of functions $\mathcal{H}$ and a pair of abstractions $\langle X, Y\rangle$, we have that Stabilize$(\mathcal{H}, X, Y)$ refines the abstraction Y to $Y_{\mathrm{stable}} \triangleq \sqcup\{Y' \mid Y' \trianglelefteq Y,\ \langle X, Y'\rangle \text{ is } \mathcal{H}\text{-complete}\}$, so that $\langle X, Y_{\mathrm{stable}}\rangle$ becomes $\mathcal{H}$-stable (i.e., $\mathcal{H}$-complete). For instance, Stabilize$(\mathcal{F}, A, B)$ minimally refines B to B' so that $\langle A, B'\rangle$ is $\mathcal{F}$-complete. Hence, while the abstraction B is refined, the abstraction A is left unchanged.

Note that if B is actually refined into $B' \lhd B$, then the $\mathcal{G}$-Stable flag is set to false so that ShellAlgo() proceeds by $\mathcal{G}$-stabilizing $\langle B', A\rangle$, i.e., by calling Stabilize$(\mathcal{G}, B, A)$. Thus, ShellAlgo$(\mathcal{F}, \mathcal{G}, A, B)$ works by iteratively refining the abstractions A and B separately, namely it refines B w.r.t. $\mathcal{F}$ while A is kept fixed and then it refines A w.r.t. $\mathcal{G}$ while B is kept fixed.

Theorem 3.1. ShellAlgo$(\mathcal{F}, \mathcal{G}, A, B) = \mathrm{Shell}_{\langle \mathcal{F}, \mathcal{G}\rangle}(\langle A, B\rangle)$.

4 Bisimulation as a Shell

Bisimulation is commonly computed by coarsest partition refinement algorithms [1,8] that iteratively refine a current partition until it becomes the bisimulation partition. Coarsest partition refinements can be cast as shells of partitions: given a property of partitions $\mathbb{P} \subseteq \mathrm{Part}(X)$, the $\mathbb{P}$-shell of $Q \in \mathrm{Part}(X)$ corresponds to the coarsest partition refinement of Q that satisfies $\mathbb{P}$, when this exists. In this section we show how bisimulation in PLTSs can be equivalently stated in terms of forward complete shells of partitions w.r.t. suitable concrete semantic functions. We also show how the above basic shell algorithm ShellAlgo() can be instantiated to compute bisimulations on PLTSs.

Shells of Partitions. Let us first recall that, given a finite set X, $\langle \mathrm{Part}(X), \preceq, \curlyvee, \curlywedge\rangle$ is a (finite) lattice where $P_1 \preceq P_2$ (i.e., P_2 is coarser than P_1 or P_1 refines P_2) iff $\forall x.P_1(x) \subseteq P_2(x)$, and its top element is $\top_{\mathrm{Part}(X)} = \{X\}$. By following the approach in [11], any partition $P \in \mathrm{Part}(X)$ can be viewed as an abstraction of $\wp(X)_{\subseteq}$ where any set $S \subseteq X$ is approximated through its minimal cover in the partition P. This is formalized by the abstract domain $\mathrm{closed}(P) \triangleq \{S \subseteq X \mid P(S) = S\}$ so that $S \in \mathrm{closed}(P)$ iff $S = \cup_{i\in I} B_i$ for some blocks $\{B_i\}_{i\in I} \subseteq P$. Note that $\varnothing, X \in \mathrm{closed}(P)$ and that $\langle \mathrm{closed}(P), \subseteq, \cup, \cap\rangle$ is a lattice. It turns out that $\langle \mathrm{closed}(P), \subseteq\rangle$ is an abstraction in $\mathrm{Abs}(\wp(X)_{\subseteq})$, where any set $S \subseteq X$ is approximated through the blocks in P covering S, namely by $\cup\{B \in P \mid B \cap S \neq \varnothing\} \in \mathrm{closed}(P)$.

The above embedding of partitions as abstract domains allows us to define a notion of forward completeness for partitions. Let $f : \wp(X) \to \wp(Y)$ be a concrete semantic function. Then, a pair of partitions $\langle P, Q\rangle \in \mathrm{Part}(X) \times \mathrm{Part}(Y)$ is (forward) f-complete when for any union $U \in \mathrm{closed}(P)$ of blocks of P, $f(U)$ is a union of blocks of Q, namely $f(U) \in \mathrm{closed}(Q)$. Also, if we additionally consider $g : \wp(Y) \to \wp(X)$ then $\langle P, Q\rangle$ is $\langle f, g\rangle$-complete when $\langle P, Q\rangle$ is f-complete and $\langle Q, P\rangle$ is g-complete. As in Section 3, forward complete shells of partitions exist. Given $\mathcal{F} \subseteq \wp(X) \to \wp(Y)$ and $\mathcal{G} \subseteq \wp(Y) \to \wp(X)$, $\mathrm{Shell}_{\langle \mathcal{F}, \mathcal{G}\rangle}(\langle P, Q\rangle)$ is the coarsest pair of partitions that (component-wise) refines the pair $\langle P, Q\rangle$ and is $\langle \mathcal{F}, \mathcal{G}\rangle$-complete, namely $\mathrm{Shell}_{\langle \mathcal{F}, \mathcal{G}\rangle}(\langle P, Q\rangle) \triangleq \curlyvee\{\langle P', Q'\rangle \in \mathrm{Part}(X) \times \mathrm{Part}(Y) \mid \langle P', Q'\rangle \preceq \langle P, Q\rangle, \langle P', Q'\rangle$ is $\langle \mathcal{F}, \mathcal{G}\rangle$-complete$\}$.

Bisimulation on PLTSs. In [11] it is shown that bisimulation on LTSs can be equivalently defined in terms of forward complete shells of partitions w.r.t. the predecessor operator. This same idea scales to the case of PLTSs taking into account that: (i) in a PLTS the target of the transition relation is a set of distributions rather than a set of

states, and (ii) bisimulation on the states of a PLTS induces an equivalence over distributions that depends on the probability that the distributions associate to the blocks of bisimilar states. Let $\mathbb{S} = \langle \Sigma, Act, \rightarrow \rangle$ be a PLTS and consider the following two functions, where $a \in Act$ and $p \in [0,1]$:

$$\mathrm{pre}_a : \wp(\mathrm{Distr}) \to \wp(\Sigma),\ \mathrm{pre}_a(D) \triangleq \{s \in \Sigma \mid s \xrightarrow{a} D\}$$
$$\mathrm{prob}_p : \wp(\Sigma) \to \wp(\mathrm{Distr}),\ \mathrm{prob}_p(S) \triangleq \{d \in \mathrm{Distr} \mid d(S) \geq p\}$$

pre_a is the a-predecessor function in the PLTS $\mathbb{S}$ while $\mathrm{prob}_p(S)$ returns the distributions whose probability on the set S is higher than p. Let $\mathrm{pre} \triangleq \{\mathrm{pre}_a\}_{a \in Act}$ and $\mathrm{prob} \triangleq \{\mathrm{prob}_p\}_{p \in [0,1]}$. It is worth noticing that this pair of sets of functions provides an encoding of the PLTS $\mathbb{S}$: pre encodes the transition relation $\rightarrow$, while any distribution d in $\mathbb{S}$ can be encoded through prob. For instance, the support of a distribution $d \in \mathrm{Distr}$ is given by the minimal set of states S such that $d \in \mathrm{prob}_1(S)$, while, for any $s \in \Sigma$, $d(s) = \sup\{p \in [0,1] \mid d \in \mathrm{prob}_p(\{s\})\}$.

Lemma 4.1. *Consider* $\langle P, \mathcal{P} \rangle \in \mathrm{Part}(\Sigma) \times \mathrm{Part}(\mathrm{Distr})$. $\langle P, \mathcal{P} \rangle$ *is* $\langle \mathrm{prob}, \mathrm{pre} \rangle$*-complete if and only if the following two conditions hold:* (i) *if* $s \xrightarrow{a} d$ *and* $t \in P(s)$ *then* $t \xrightarrow{a} \mathcal{P}(d)$*;* (ii) *if* $e \in \mathcal{P}(d)$ *then* $d \equiv_P e$.

Consequently, a partition $P \in \mathrm{Part}(\Sigma)$ is a bisimulation on $\mathbb{S}$ if and only if $\langle P, \equiv_P \rangle$ is $\langle \mathrm{prob}, \mathrm{pre} \rangle$-complete. In turn, the coarsest bisimulation P_{bis} on $\mathbb{S}$ can be obtained as a forward complete shell of partitions.

Theorem 4.2. $\langle P_{\mathrm{bis}}, \equiv_{P_{\mathrm{bis}}} \rangle = \mathrm{Shell}_{\langle \mathrm{prob}, \mathrm{pre} \rangle}(\top_{\mathrm{Part}(\Sigma)}, \top_{\mathrm{Part}(\mathrm{Distr})})$.

Bisimulation Algorithm. By Theorem 4.2, P_{bis} can be computed as a partition shell by instantiating the basic shell algorithm in Figure 1 to $\mathcal{F} = \{\mathrm{prob}_p\}_{p \in [0,1]}$ and $\mathcal{G} = \{\mathrm{pre}_a\}_{a \in Act}$, and by viewing partitions in $\mathrm{Part}(\Sigma) \times \mathrm{Part}(\mathrm{Distr})$ as abstract domains. This leads to design a bisimulation algorithm called PBis that maintains a pair of state and distribution partitions $\langle P, \mathcal{P} \rangle \in \mathrm{Part}(\Sigma) \times \mathrm{Part}(\mathrm{Distr})$ and whose initialization and stabilization functions are given in Figure 2.

The function call preStabilize($\langle \mathcal{P}, P \rangle$) refines the state partition P into P' so that $\langle \mathcal{P}, P' \rangle$ is pre-complete. Note (cf. Lemma 4.1) that in order to get pre-completeness it is sufficient to minimally refine P so that for any block of distributions $C \in \mathcal{P}$, and for any incoming label $a \in \mathrm{in}(C)$, $\mathrm{pre}_a(C)$ is a union of blocks of P. If $\mathrm{pre}_a(C)$ is not a union of blocks of P then $\mathrm{pre}_a(C) \subseteq \Sigma$ is called a splitter of P, and we denote by $\mathrm{Split}(P, \mathrm{pre}_a(C))$ the partition obtained from P by replacing each block $B \in P$ with the nonempty sets $B \cap \mathrm{pre}_a(C)$ and $B \smallsetminus \mathrm{pre}_a(C)$. Notice that when some $\mathrm{pre}_a(C)$ is already a union of blocks of P we have that $\mathrm{Split}(P, \mathrm{pre}_a(C)) = P$, i.e., we also allow no splitting. Hence, preStabilize() refines P by iteratively splitting P w.r.t. $\mathrm{pre}_a(C)$, for all $C \in \mathcal{P}$ and $a \in \mathrm{in}(C)$. On the other hand, the function call probStabilize($\langle P, \mathcal{P} \rangle$) refines the current distribution partition $\mathcal{P}$ into $\mathcal{P}'$ so that $\langle P, \mathcal{P}' \rangle$ is prob-complete. It turns out that $\langle P, \mathcal{P} \rangle$ is prob-complete when for any block $B \in P$ and any distribution $d \in \mathrm{Distr}$, $\{e \in \mathrm{Distr} \mid e(B) = d(B)\}$ is a union of blocks of $\mathcal{P}$. Thus, probStabilize() iteratively splits the distribution partition $\mathcal{P}$ w.r.t. $\{e \in \mathrm{Distr} \mid e(B) = d(B)\}$, for all $B \in P$ and $d \in \mathrm{Distr}$.

```
Initialize() {
  forall s ∈ Σ do P(s) := Σ; forall d ∈ Distr do 𝒫(d) := Distr;
  preStabilize(⟨𝒫, P⟩); preStable := probStabilize(⟨P, 𝒫⟩); probStable := true;
}
bool preStabilize(⟨𝒫, P⟩) {
  P_old := P;
  forall C ∈ 𝒫 do  forall a ∈ in(C) do  P := Split(P, pre_a(C));
  return (P ≠ P_old)
}
bool probStabilize(⟨P, 𝒫⟩) {
  𝒫_old := 𝒫;
  forall B ∈ P do  forall d ∈ Distr do  𝒫 := Split(𝒫, {e ∈ Distr | e(B) = d(B)});
  return (𝒫 ≠ 𝒫_old)
}
```

Fig. 2. Bisimulation Algorithm PBis

Theorem 4.3. *For a finite PLTS* $\mathcal{S}$, $\mathrm{PBis}(\mathcal{S})$ *terminates and is correct, i.e., if* $\langle P, \mathcal{P}\rangle$ *is the output of* $\mathrm{PBis}(\mathcal{S})$ *then* $P = P_{\mathrm{bis}}$ *and* $\mathcal{P} = \equiv_{P_{\mathrm{bis}}}$.

Implementation. Baier-Engelen-Majster's two-phased partitioning algorithm [1] is the standard procedure for computing the bisimulation P_{bis}. This bisimulation algorithm can be essentially viewed as an implementation of the above PBis algorithm, since the two phases of Baier et al.'s algorithm (see [1, Figure 9]) coincide with our preStabilize() and probStabilize() functions. The only remarkable difference is that instead of using a single partition over all the distributions in Distr, Baier et al.'s algorithm maintains a so-called step partition, namely, a family of partitions $\{M_a\}_{a\in Act}$ such that, for any $a \in Act$, M_a is a partition of the distributions in $\mathrm{post}_a(\Sigma)$, i.e., the distributions that have an incoming edge labeled with a. As a consequence, in the phase that corresponds to probStabilize(), any partition M_a is split w.r.t. all the splitters $\{e \in \mathrm{post}_a(\Sigma) \mid e(B) = d(B)\}$, where $B \in P$ and $d \in \mathrm{post}_a(\Sigma)$. Baier et al.'s algorithm is implemented by exploiting Hopcroft's "process the smaller half" principle when splitting the state partition w.r.t. a splitter $\mathrm{pre}_a(C)$ and this allows to obtain a procedure that computes bisimulation in $O(|{\to}||\Sigma|(\log|{\to}| + \log|\Sigma|))$ time and $O(|{\to}||\Sigma|)$ space.

5 Simulation as a Shell

Shells of Preorders. Recall that, given any finite set X, $\langle \mathrm{PreOrd}(X), \subseteq, \cup^t, \cap\rangle$ is a lattice, where $R_1 \cup^t R_2$ is the transitive closure of $R_1 \cup R_2$ and the top element is $\top_{\mathrm{PreOrd}(X)} \triangleq X \times X$. Analogously to partitions, any preorder $R \in \mathrm{PreOrd}(X)$ can be viewed as an abstraction of $\wp(X)_{\subseteq}$, where any set $S \subseteq X$ is approximated by its R-closure $R(S)$. Formally, a preorder $R \in \mathrm{PreOrd}(X)$ can be viewed as the abstract domain $\mathrm{closed}(R) \triangleq \{S \subseteq X \mid R(S) = S\}$. Observe that $S \in \mathrm{closed}(R)$ iff $S = \cup_{i\in I} R(x_i)$ for some set $\{x_i\}_{i\in I} \subseteq X$ and that $\langle \mathrm{closed}(R), \subseteq, \cup, \cap\rangle$ is a lattice

(note that $\varnothing, X \in \mathrm{closed}(R)$). It turns out that $\mathrm{closed}(R) \in \mathrm{Abs}(\wp(X)_{\subseteq})$: this means that any set $S \subseteq X$ is approximated by its R-closure, namely by $R(S) \in \mathrm{closed}(R)$.

Given the functions $\langle \mathcal{F}, \mathcal{G} \rangle \subseteq (\wp(X) \to \wp(Y)) \times (\wp(Y) \to \wp(X))$, a pair of preorders $\langle R, S \rangle \in \mathrm{PreOrd}(X) \times \mathrm{PreOrd}(Y)$ is (forward) $\langle \mathcal{F}, \mathcal{G} \rangle$-complete when for any $f \in \mathcal{F}$ and $g \in \mathcal{G}$, if $\langle U, V \rangle \in \mathrm{closed}(R) \times \mathrm{closed}(S)$ then $\langle f(U), g(V) \rangle \in \mathrm{closed}(S) \times \mathrm{closed}(R)$. Forward complete shells of preorders are therefore defined as follows: $\mathrm{Shell}_{\langle \mathcal{F}, \mathcal{G} \rangle}(\langle R, S \rangle)$ is the largest pair of preorders $\langle R', S' \rangle \subseteq \langle R, S \rangle$ which is $\langle \mathcal{F}, \mathcal{G} \rangle$-complete.

Simulation on PLTSs. Similarly to the case of bisimulation, simulation can be equivalently expressed in terms of forward completeness w.r.t. $\mathrm{prob} = \{\mathrm{prob}_p\}_{p \in [0,1]}$ and $\mathrm{pre} = \{\mathrm{pre}_a\}_{a \in Act}$.

Lemma 5.1. *Let $\langle R, \mathcal{R} \rangle \in \mathrm{PreOrd}(\Sigma) \times \mathrm{PreOrd}(\mathrm{Distr})$. Then, $\langle R, \mathcal{R} \rangle$ is $\langle \mathrm{prob}, \mathrm{pre} \rangle$-complete if and only if the following two conditions hold:* (i) *if $t \in R(s)$ and $s \xrightarrow{a} d$ then there exists e such that $t \xrightarrow{a} e$ and $e \in \mathcal{R}(d)$;* (ii) *if $e \in \mathcal{R}(d)$ then $d \leq_R e$.*

Thus, a preorder $R \in \mathrm{PreOrd}(\Sigma)$ is a simulation on $\mathbb{S}$ if and only if $\langle R, \leq_R \rangle$ is $\langle \mathrm{prob}, \mathrm{pre} \rangle$-complete. In turn, the greatest simulation preorder R_{sim} can be obtained as a preorder shell.

Theorem 5.2. $\langle R_{\mathrm{sim}}, \equiv_{R_{\mathrm{sim}}} \rangle = \mathrm{Shell}_{\langle \mathrm{prob}, \mathrm{pre} \rangle}(\top_{\mathrm{PreOrd}(\Sigma)}, \top_{\mathrm{PreOrd}(\mathrm{Distr})})$.

6 A New Efficient Probabilistic Simulation Algorithm

A new efficient algorithm for computing simulations in PLTSs, called PSim, is designed in this section by instantiating the basic shell algorithm to $\mathcal{F} = \{\mathrm{prob}_p\}_{p \in [0,1]}$ and $\mathcal{G} = \{\mathrm{pre}_a\}_{a \in Act}$, and by viewing preorders in $\mathrm{PreOrd}(\Sigma) \times \mathrm{PreOrd}(\mathrm{Distr})$ as abstract domains.

The high-level design of PSim is that of ShellAlgo in Figure 1, the only difference being that the input is a PLTS $\mathbb{S}$. PSim maintains a pair of state and distribution preorders $\langle R, \mathcal{R} \rangle \in \mathrm{PreOrd}(\Sigma) \times \mathrm{PreOrd}(\mathrm{Distr})$, whose initialization and stabilization functions are given in Figure 3 and 4.

The function preStabilize() makes the pair $\langle \mathcal{R}, R \rangle$ pre-complete by refining the state preorder R until there exists a transition $s \xrightarrow{a} d$ such that and $R(s) \not\subseteq \mathrm{pre}_a(\mathcal{R}(d))$. Such a refinement can be efficiently done by following the incremental approach of Henzinger et al. [6] for nonprobabilistic LTSs. On the other hand, the function probStabilize() makes the pair $\langle \mathcal{R}, R \rangle$ prob-complete by refining the distribution preorder $\mathcal{R}$ by iteratively refining it until there exist e, d such that $e \in \mathcal{R}(d)$ and $d \not\leq_R e$. Here, in order to get an efficient incremental computation, we resort to the approach of Zhang et al.'s simulation algorithm [13], and we stabilize the distribution preorder $\mathcal{R}$ by computing sequences of maximum flow problems. More precisely, given a pair of distributions (d, e), successive calls to probStabilize() might repeatedly check whether $d \leq_R e$ where R is the current (new) state preorder. Let us recall [1] that the test $d \leq_R e$ can be implemented by checking whether the maximum flow over a network built out of (d, e) and R, here denoted by $\mathcal{N}(d, e, R)$, is 1. Zhang et al. [13] observe that the networks for

```
1  Initialize() {
     // Initialize R and ℛ
2    forall s ∈ Σ do R(s) := {t ∈ Σ | out(s) ⊆ out(t)};
3    forall d ∈ Distr do ℛ(d) := {e ∈ Distr | Init_SMF(d, e, R) = true};
     // Initialize in
4    forall d ∈ Distr do in(d) := {a ∈ Act | pre_a(d) ≠ ∅};
     // Initialize Count
5    forall e ∈ Distr do
6      forall a ∈ in(e) do
7        forall x ∈ pre_a(Distr) do
8          Count(x, a, e) := |{d ∈ Distr | x -a-> d, d ∈ ℛ(e), a ∈ in(e)}|;
     // Initialize Remove
9    forall d ∈ Distr do
10     forall a ∈ in(d) do
11       Remove_a(d) := {s ∈ Σ | a ∈ out(s), s -a->> ℛ(d)};
     // Initialize Stability Flags
12   probStable := true;
13   if ∃e ∈ Distr, a ∈ in(e) such that Remove_a(e) ≠ ∅ then preStable := false;
14   else preStable := true;
     // Initialize Listener
15   forall x, y ∈ Σ do Listener(x, y) := {(d, e) | x ∈ supp(d), e ∈ supp(e)};
     // Initialize Deleted Arcs
16   Deleted := ∅;
17 }
```

Fig. 3. Initialization function

a given pair (d, e) across successive iterations are very similar, since they differ only by deletion of some edges due to the refinement of R. Therefore, in order to incrementally deal with this sequence of tests, Zhang et al.'s algorithm saves after each iteration the current network $\mathcal{N}(d, e, R)$ together with its maximum flow information, and this allows us to use at the next iteration a so-called preflow algorithm which is initialized with the previous maximum flow function. Due to lack of space, we do not discuss the details of the preflow algorithm in [13], that is used here as a black box that efficiently solves the sequence of maximum flow problems that arise for a same network.

PSim is designed around the following data structures. First, the two preorders $R \subseteq \Sigma \times \Sigma$ and $\mathcal{R} \subseteq \text{Distr} \times \text{Distr}$, which are stored as boolean matrices and are initialized so that they are coarser than R_{sim} and $\leq_{R_{\text{sim}}}$. In particular, the initial preorder R is coarser than R_{sim} since if $s \xrightarrow{a} d$ and $t \not\xrightarrow{a}$ then $t \notin R_{\text{sim}}(s)$. Moreover, line 3 initializes $\mathcal{R}$ so that $\mathcal{R} = \leq_R$: this is done by calling the function Init_SMF(d, e, R) which in turn calls the preflow algorithm to check whether $d \leq_R e$, and in case this is true, stores the network $\mathcal{N}(d, e, R)$ to reuse it in later calls to probStabilize(). The additional data structures used by PSim come from the efficient refinement approaches used in [6] and [13]. Indeed, as in [6], for any distribution e and for any incoming action $a \in \text{in}(e)$, we store and maintain a set $\text{Remove}_a(e) \triangleq \{s \in \Sigma \mid s \xrightarrow{a}, s \xrightarrow{a}\!\!\!\!\rightarrow \mathcal{R}(e)\}$ that is

```
1  bool preStabilize(⟨ℛ, R⟩) {
2    Deleted := ∅;
3    while ∃ Remove_a(e) ≠ ∅ do
4      Remove := Remove_a(e); Remove_a(e) := ∅;
5      forall t -a-> e do
6        forall w ∈ Remove do
7          if w ∈ R(t) then Deleted := Deleted ∪ {(t, w)}; R(t) := R(t) \ {w};
8    if (Deleted ≠ ∅) then probStable := false;
9    return probStable;
10 }

1  bool probStabilize(⟨R, ℛ⟩) {
2    forall (t, w) ∈ Deleted do
3      forall (d, e) ∈ Listener(t, w) such that e ∈ ℛ(d) do
4        if SMF(d, e, (t, w)) = false then
5          ℛ(d) := ℛ(d) ∖ {e};
6          forall b ∈ in(e) ∩ in(d) do
7            forall s -b-> e do
8              Count(s, b, d)- -;
9              if Count(s, b, d) = 0 then
10               Remove_b(d) := Remove_b(d) ∪ {s}; preStable := false;
11   return preStable;
12 }
```

Fig. 4. Stabilization functions

used to refine the relation R such that if $t \xrightarrow{a} e$ then $R(t)$ is pruned to $R(t) \setminus \mathrm{Remove}_a(e)$ (lines 5-7 of preStabilize()). The Count table is used to efficiently refill the remove sets (line 10 of probStabilize()), since it allows to test whether $s \xrightarrow{a} \mathcal{R}(e)$ in $O(1)$. On the other hand, in order to get an efficient refinement also for the distribution preorder $\mathcal{R}$, as in Zhang et al. [13], for any pair of states (x, y) we compute and store a set $\mathrm{Listener}(x, y) \triangleq \{(d, e) \in \mathrm{Distr} \times \mathrm{Distr} \mid x \in \mathrm{supp}(d),\ y \in \mathrm{supp}(e)\}$ that contains all the pairs of distributions (d, e) such that the network $\mathcal{N}(d, e, R)$ could contain the edge (x, y), i.e., the networks that are affected when the pair (x, y) is removed from R. Indeed, these sets are used in probStabilize() to recognize the pairs (d, e) that have been affected by the refinement of R due to the previous call of preStabilize() (lines 2-3 of probStabilize()).

As a result, at the end of the initialization, the probStable flag is true (due to the initialization of $\mathcal{R}$ as $\leq_R$), whereas the preStable flag is false if there is at least a nonempty remove set. The main loop of PSim then proceeds by repeatedly calling the stabilization functions until $\langle R, \mathcal{R} \rangle$ becomes $\langle \mathrm{prob}, \mathrm{pre} \rangle$-complete. More precisely, a call to preStabilize(): (i) empties all the Remove sets, (ii) collects all the pairs removed from R into the set $\mathrm{Deleted}$, and (iii) sets the probStable flag to false when R has changed. On the other hand, a call to probStabilize() relies on the sets $\mathrm{Deleted}$ and $\mathrm{Listener}$ to identify

all the networks $\mathcal{N}(d, e, R)$ that have been affected by the refinement of R due to preStabilize(). For any pair (t, w) that has been removed from R, the call SMF$(d, e, (t, w))$ at line 4 removes the edge (t, w) from the network for (d, e) and checks whether it still has a maximum flow equals to 1. Hence, if this is not the case, e is removed from $\mathcal{R}(d)$. Notice that such a pruning may induce an update of some $\mathrm{Remove}_b(d)$ set, which in turn triggers a further call of preStabilize() by setting the preStable flag to false.

The correctness of PSim is a direct consequence of Theorems 3.1 and 5.2 and the fact that the procedures in Figure 4 correctly stabilize the preorders R and $\mathcal{R}$. The complexity bounds of PSim are given in terms of the following sizes. Let $\mathcal{S} = \langle \Sigma, Act, \rightarrow \rangle$ be the input PLTS. Then, $|\mathrm{Distr}| = |\bigcup_{a \in Act} \mathrm{post}_a(\Sigma)|$ is the number of distributions appearing as target of some transition. Also, the number of edges in $\mathcal{S}$ is $|\rightarrow| \leq |\Sigma||\mathrm{Distr}||Act|$. Moreover, we consider the following sizes: $p \triangleq \sum_{d \in \mathrm{Distr}} |\mathrm{supp}(d)|$ and $m \triangleq \sum_{a \in Act} \sum_{s \in \Sigma} \sum_{d \in \mathrm{post}_a(s)} (|\mathrm{supp}(d)| + 1)$. Thus, p represents the full size of $\mathrm{post}(\Sigma)$, being the number of states that appear in the support of some distribution in $\mathrm{post}(\Sigma)$, while m represents the number of transitions from states to states in $\mathcal{S}$, where a "state transition" (s, t) is taken into account when $s \xrightarrow{a} d$ and $t \in \mathrm{supp}(d)$. Clearly, $|\mathrm{Distr}| \leq p \leq |\mathrm{Distr}||\Sigma|$ and $|\Sigma| \leq |\rightarrow| \leq m \leq |\mathrm{Distr}||\Sigma|$. The key point to remark is that $p \leq m$, since the "states" of $\mathcal{S}$ are always less than the "state transitions" in $\mathcal{S}$.

Theorem 6.1 (Correctness and Complexity). *Let $\mathcal{S}$ be a finite PLTS. $\mathrm{PSim}(\mathcal{S})$ terminates with output $\langle R_{\mathrm{sim}}, \leq_{R_{\mathrm{sim}}} \rangle$ and runs in $O(|\Sigma|(p^2 + |\rightarrow|))$-time and $O(p^2 + (|\Sigma| + |\mathrm{Distr}|)|\rightarrow|)$-space.*

It is easy to observe that $(|\Sigma| + |\mathrm{Distr}|)|\rightarrow| \leq m^2$, so that PSim results to be more efficient than the most efficient probabilistic simulation algorithm in literature, that is Zhang et al.'s algorithm [13], that runs in $O(|\Sigma|m^2)$-time and $O(m^2)$-space. Our scaling down from the factor m to p, that is from the size of the "state transitions" to the size of the "state" space, basically depends on the fact that in Zhang et al.'s algorithm the same test $d \not\leq_R e$ is repeated for every pair of states (s_i, t_i) such that $s_i \in \mathrm{pre}_a(d), t_i \in \mathrm{pre}_a(e)$, whereas in PSim once the test $d \not\leq_R e$ has been performed, every state t_i is removed from $R(s_i)$. Such a difference becomes evident when the input PLTS $\mathcal{S}$ degenerates to a LTS. In this case a call to the function SMF() can be executed in $O(1)$, so that the time complexity of [13] boils down to $O(|\rightarrow|^2)$, whereas in this case PSim runs in $O(|\Sigma||\rightarrow|)$-time, essentially reducing to Henzinger et al. [6]'s non-probabilistic simulation algorithm. As a further difference, it is worth observing that Zhang et al.'s algorithm relies on a positive logic that at each iteration i computes the pairs (s_i, t_i) such that $t_i \in R_i(s_i)$, whereas PSim follows a dual, negative, strategy that removes from R_i the pairs (s_i, t_i) such that $t_i \notin R_i(s_i)$.

7 Future Work

We have shown how abstract interpretation can be applied in the context of behavioral relations between probabilistic processes. We focused here on bisimulation/simulation relations on PLTSs and we showed how efficient algorithms that compute these behavioral relations can be derived. As future work, we plan to investigate how this abstract

interpretation approach can be adapted to characterize the weak variants of bisimulation/simulation and the so-called probabilistic bisimulations/simulations on PLTSs [12]. We also intend to apply a coarsest partition refinement approach to design a "symbolic" version of our PSim simulation algorithm. Analogously to the symbolic algorithm by Ranzato and Tapparo [10] for nonprobabilistic simulation, the basic idea is to symbolically represent the relations R on states and $\mathcal{R}$ on distributions through partitions (of states and distributions) and corresponding relations between their blocks. It is worth noting that this partition refinement approach has been already applied by Zhang [14] to design a space-efficient simulation algorithm for PLTSs. Finally, we envisage to study how the abstract interpretation approach can be related to the logical characterizations of behavioral relations of probabilistic processes studied e.g. in [9].

References

1. Baier, C., Engelen, B., Majster-Cederbaum, M.: Deciding bisimilarity and similarity for probabilistic processes. J. Comp. Syst. Sci. 60, 187–231 (2000)
2. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. 4th ACM POPL, pp. 238–252 (1977)
3. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Proc. 6th ACM POPL, pp. 269–282 (1979)
4. Giacobazzi, R., Quintarelli, E.: Incompleteness, counterexamples, and refinements in abstract model-checking. In: Cousot, P. (ed.) SAS 2001. LNCS, vol. 2126, pp. 356–373. Springer, Heidelberg (2001)
5. van Glabbeek, R.J., Smolka, S., Steffen, B., Tofts, C.: Reactive, generative and stratified models for probabilistic processes. In: Proc. IEEE LICS 1990, pp. 130–141 (1990)
6. Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing simulations on finite and infinite graphs. In: Proc. 36th FOCS, pp. 453–462 (1995)
7. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Information and Computation 94(1), 1–28 (1991)
8. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM J. Comput. 16(6), 973–989 (1987)
9. Parma, A., Segala, R.: Logical characterizations of bisimulations for discrete probabilistic systems. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 287–301. Springer, Heidelberg (2007)
10. Ranzato, F., Tapparo, F.: A new efficient simulation equivalence algorithm. In: Proc. IEEE LICS 2007, pp. 171–180 (2007)
11. Ranzato, F., Tapparo, F.: Generalized strong preservation by abstract interpretation. J. Logic and Computation 17(1), 157–197 (2007)
12. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nordic J. of Computing 2(2), 250–273 (1995)
13. Zhang, L., Hermanns, H., Eisenbrand, F., Jansen, D.N.: Flow faster: efficient decision algorithms for probabilistic simulations. Logical Methods in Computer Science 4(4) (2008)
14. Zhang, L.: A space-efficient probabilistic simulation algorithm. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 248–263. Springer, Heidelberg (2008)

On the Semantics of Markov Automata

Yuxin Deng[1,3,4,⋆] and Matthew Hennessy[2,⋆⋆]

[1] Shanghai Jiao Tong University
[2] Trinity College Dublin
[3] Carnegie Mellon University
[4] Chinese Academy of Sciences

Abstract. Markov automata describe systems in terms of events which may be nondeterministic, may occur probabilistically, or may be subject to time delays. We define a novel notion of weak bisimulation for such systems and prove that this provides both a sound and complete proof methodology for a natural extensional behavioural equivalence between such systems, a generalisation of *reduction barbed congruence*, the well-known touchstone equivalence for a large variety of process description languages.

1 Introduction

Markov Automata (MA) [8] describe system behaviour in terms of non-deterministic, probabilistic and timed events. The first two kinds of events are well-known from Probabilistic Automata (PA) [22,23] and Probabilistic Labelled Transition Systems (pLTSs) [5], while the third are taken to be random delays, governed by negative exponential distributions parametrised by some delay $\lambda \in \mathbb{R}^+$. As explained in [10], timed events can be given a straightforward operational semantics in terms of their parametric delays.

For example, consider the MAs in Figure 1, taken from [8]. From the initial state of the first automaton, s, there is a race between two possible timed events, denoted by double-headed arrows, each governed by the same rate, 4λ, for some arbitrary $\lambda \in \mathbb{R}^+$. If one of these events wins, the state of the automaton changes to s_a, from which some external action a can happen. If the other timed event wins, the change of state is to s_1, from which an internal unobservable action, denoted by τ, can occur. Moreover, the effect of this internal action is probabilistic; fifty percent of the time the state change will be to s_b, where action b can occur, while with the same probability the change will be to s_c, where c can occur. Formally this probabilistic behaviour is represented as an action from a state, such as s_1, to a distribution over states, represented as a darkened circle connected to states in the support of the distribution, labelled with their probabilities.

On the other hand the second automaton is much more straightforward. From its initial state there is a race between three timed events, two running at the same rate and one at double the rate. Then one of the (external) actions a, b, c occurs depending on which event wins the race.

⋆ Partially supported by the Natural Science Foundation of China under Grant No. 61033002, the Qatar National Research Fund under grant NPRP 09-1107-1-168, and the Opening Fund of Top Key Discipline of Computer Software and Theory in Zhejiang Provincial Colleges at Zhejiang Normal University.

⋆⋆ Supported financially by SFI project no. SFI 06 IN.1 1898.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 307–318, 2011.

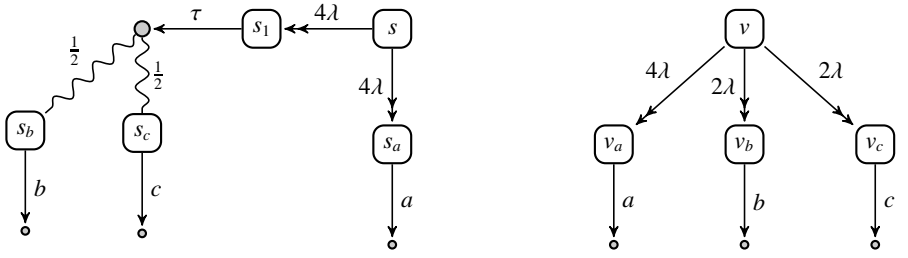


Fig. 1. Timed transitions and distributions

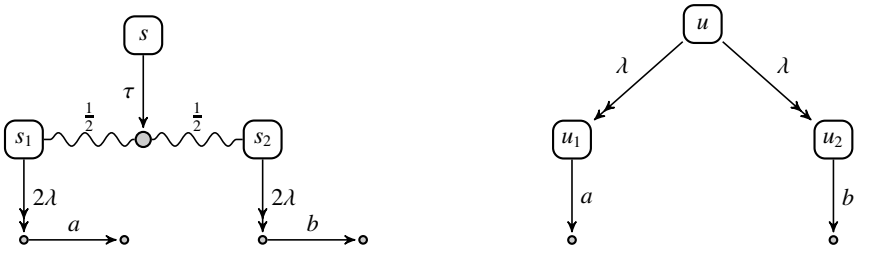


Fig. 2. Timed transitions and distributions, again

Providing a satisfactory behavioural model of MAs is necessarily a complicated undertaking. But as pointed out in [8], because of the nature of their underlying distributions, the timed events can be satisfactorily explained in terms of simple probabilistic distributions determined by their rates. They propose a translation of MAs into PAs (cf. Section 2). Since behavioural theories have already been developed for PAs [24,6,18,4], we therefore automatically obtain such theories for MAs, via their induced PAs.

However, if one uses a standard behavioural theory for PAs, such as weak bisimulation equivalence as defined in [15,24,18,6] then the two MAs in Figure 1 are distinguished. Instead the authors of [8] propose a new bisimulation equivalence between PAs, which enjoys standard properties such as compositionality, and which identifies these two MAs. But as the authors point out their equivalence still distinguishes between the MAs in Figure 2. The question naturally arises: which MAs should be distinguished, and which be deemed equivalent. This is the topic of the current paper.

We approach the question indirectly, by giving criteria for reasonable behavioural equivalences between MAs; this induces a *touchstone* extensional equivalence between systems, namely the largest equivalence, $\approx_{\text{behav}}$, which satisfies these criteria. Thus two MAs should only be distinguished on the basis of the chosen criteria.

Having an independent notion of which systems should, and which should not, be distinguished, one can then justify a particular notion of bisimulation by showing that it captures precisely the touchstone equivalence, $\approx_{\text{behav}}$. In other words, a particular definition of bisimulation is appropriate because $\approx_{bis}$, the associated bisimulation equivalence,

(i) is *sound* w.r.t. the touchstone equivalence, that is $s_1 \approx_{bis} s_2$ implies $s_1 \approx_{behav} s_2$
(ii) provides a *complete* proof methodology for the touchstone equivalence, that is $s_1 \approx_{behav} s_2$ implies $s_1 \approx_{bis} s_2$.

This approach originated in [13] but has now been widely used for different process description languages; for example see [14,20] for its application to higher-order process languages, [19] for mobile ambients and [9] for asynchronous languages. Moreover in each case the distinguishing criteria are more or less the same. The touchstone equivalence should be

(i) *compositional*; that is preserved by natural operators for constructing systems
(ii) *barb-preserving*; barbs are simple experiments observers perform on systems
(iii) *reduction-closed*; this is a natural condition on the reduction semantics of systems which ensures that nondeterministic choices are in some sense preserved.

We adapt this approach to MAs. Using natural versions of these criteria for MAs we obtain an appropriate touchstone equivalence, *reduction barbed congruence* ($\approx_{rbc}$). We then develop a new theory of bisimulations which is both sound and complete for $\approx_{rbc}$.

The remainder of the paper is organised as follows. In the next section we give our definition of Markov automata MA, a slight generalisation of that in [8]; in addition to the timed events parametrised on specific delays, we have special timed events which have indefinite, or imprecise delay times associated with them. In order to model the delay operators probabilistically, we then show how to translate an MA into a PA, as suggested in [8]. For this purpose we use a slight variation, called MLTSs, in which there are distinguished actions labelled by weights. We then develop our new definition of bisimulation equivalence for MLTSs, thereby inducing bisimulation equivalence between MAs; this construction is illustrated via examples. In Section 3 we show how MAs can be composed, using a parallel operator based on CCS [17]. In fact this is extended to an interpretation of an Markovian extension of CCS, mCCS, as an MA. We then show that bisimulation equivalence is preserved by this form of composition.

Section 4 contains the main theoretical results of the paper. We give a formal definition of the touchstone equivalence $\approx_{rbc}$, and outline the proof that this is captured precisely by our new notion of bisimulation. The paper ends with a brief comparison with related work in Section 5.

2 Markov Automata

A (discrete) probability subdistribution over a set S is a function $\Delta : S \to [0,1]$ with $\sum_{s \in S} \Delta(s) \leq 1$; the support of such an Δ is the set $\lceil \Delta \rceil = \{ s \in S \mid \Delta(s) > 0 \}$. A subdistribution is a (total, or full) distribution if $\sum_{s \in S} \Delta(s) = 1$. The point distribution $\overline{s}$ assigns probability 1 to s and 0 to all other elements of S, so that $\lceil \overline{s} \rceil = s$. We use $\mathcal{D}_{sub}(S)$ to denote the set of subdistributions over S, and $\mathcal{D}(S)$ its subset of full distributions.

We write $\mathbb{R}^+$ for the set of all positive real numbers. Let $\{\Delta_k \mid k \in K\}$ be a set of subdistributions, possibly infinite. Then $\sum_{k \in K} \Delta_k$ is the real-valued function in $S \to \mathbb{R}^+$ defined by $(\sum_{k \in K} \Delta_k)(s) := \sum_{k \in K} \Delta_k(s)$. This is a partial operation on subdistributions because for some state s the sum of $\Delta_k(s)$ might exceed 1. If the index set is finite, say $\{1..n\}$, we often write $\Delta_1 + \ldots + \Delta_n$. For p a real number from $[0,1]$ we use $p \cdot \Delta$ to denote the subdistribution given by $(p \cdot \Delta)(s) := p \cdot \Delta(s)$. If $\sum_{k \in K} p_k = 1$ for some collection of $p_k \geq 0$, and the Δ_k are distributions, then so is $\sum_{k \in K} p_k \cdot \Delta_k$.

Definition 1. *A* Markov automaton *(MA), is a quadruple* $\langle S, \mathsf{Act}_\tau, \rightarrow, \mapsto \rangle$*, where*

(i) S *is a set of states*
(ii) Act_τ *is a set of transition labels, with distinguished element* τ
(iii) the relation $\rightarrow$ *is a subset of* $S \times \mathsf{Act}_\tau \times \mathcal{D}(S)$
(iv) the relation $\mapsto$ *is a subset of* $S \times (\mathbb{R}^+ \cup \{\delta\}) \times \mathcal{D}(S)$

satisfying (a) $s \overset{\boldsymbol{d}}{\mapsto} \Delta$ *implies* $s \overset{\tau}{\not\longrightarrow}$, $\boldsymbol{d} = \delta$ *or* λ*, (b)* $s \overset{\delta}{\mapsto} \Delta_1$ *and* $s \overset{\delta}{\mapsto} \Delta_2$ *implies* $\Delta_1 = \Delta_2$*. The MA is* finitary *if* S *is finite and each state has only finitely many outgoing transitions.*

This is a mild generalisation of the MAs in [8]; for example we allow the residual of a timed action to be a distribution, and *maximal progress*, assumption (a), is built in to the definition. But the major extension is the introduction of the indefinite delay actions denoted by the special action δ, $s \overset{\delta}{\mapsto} \Delta$; this can be viewed as a timed action whose underlying rate is unknown. Such indefinite actions, often called *passive* when they are external, are widely used in the literature [3,12], although their precise properties vary between publications; see [10], page 66 for a discussion.

Following [8], we study MAs indirectly, by considering their derived structures.

Definition 2. *A* Markov labelled transition system *(MLTS) is a triple* $\langle S, \mathsf{Act}_\tau, \rightarrow \rangle$*, where* S *and* Act_τ *are as in Definition 1, and* $\rightarrow$ *is a subset of* $S \times (\mathsf{Act}_{\tau,\delta} \cup \mathbb{R}^+) \times \mathcal{D}(S)$ *satisfying* $s \xrightarrow{\lambda_1} \Delta_1$ *and* $s \xrightarrow{\lambda_2} \Delta_2$ *implies* $\lambda_1 = \lambda_2$ *and* $\Delta_1 = \Delta_2$*, in addition to the constraints corresponding to (a) and (b) in Definition 1. Here* $\mathsf{Act}_{\tau,\delta}$ *means* $\mathsf{Act}_\tau \cup \{\delta\}$*.* □

A (non-probabilistic) labelled transition system (LTS) may be viewed as a degenerate MLTS — one in which only point distributions are used, and the special actions labelled by δ and $\lambda \in \mathbb{R}^+$ are vacuous. An MLTS is *finitary* if the state set S is finite and for each $s \in S$ the set $\{(\mu, \Delta) \mid s \xrightarrow{\mu} \Delta,\ \mu \in \mathsf{Act}_{\tau,\delta} \cup \mathbb{R}^+,\ \Delta \in \mathcal{D}(S)\}$ is finite.

Admittedly MAs and MLTSs are very similar; the difference lies in the intent. We interpret the former in the latter, thereby modelling the passage of time probabilistically. The essential ingredient in the interpretation is the function on the states of an MA, defined by $\mathsf{Rate}(s) = \sum\{\lambda_i \mid s \overset{\lambda_i}{\mapsto} \Delta_i\}$. Given an MA M as in Definition 1 the MLTS $\mathsf{mlts}(M)$ is given by $\langle S, \mathsf{Act}_\tau, \rightarrow \rangle$ where:

(a) for $\mu \in \mathsf{Act}_\tau$ the actions $s \xrightarrow{\mu} \Delta$ are inherited from M
(b) $s \xrightarrow{\delta} \Delta$ whenever $s \overset{\delta}{\mapsto} \Delta$ in M
(c) for $\lambda \in \mathbb{R}^+$, $s \xrightarrow{\lambda} \Delta$ if $\mathsf{Rate}(s) = \lambda > 0$ and $\Delta = \sum\{p_i \cdot \Delta_i \mid s \overset{\lambda_i}{\mapsto} \Delta_i\}$ where $p_i = \frac{\lambda_i}{\mathsf{Rate}(s)}$

Example 1. The derived MLTSs of the two MAs in Figure 1 are given in Figure 3. Note that the time dependent race between the evolution of s to s_a or s_1 in Figure 1 is represented in Figure 3 by a single arrow labelled by the total rate of s to a distribution representing the chances of s_1 and s_2 winning the race. Similarly in the second MA the race from v to v_a, v_b, v_c is now represented by a single weighted arrow to a similar distribution. The weights on these arrows will be used for compositional reasoning. □

In an MLTS actions are only performed by states, but in general we allow distributions over states to perform an action. For this purpose, we *lift* these relations so that they also apply to subdistributions [5].

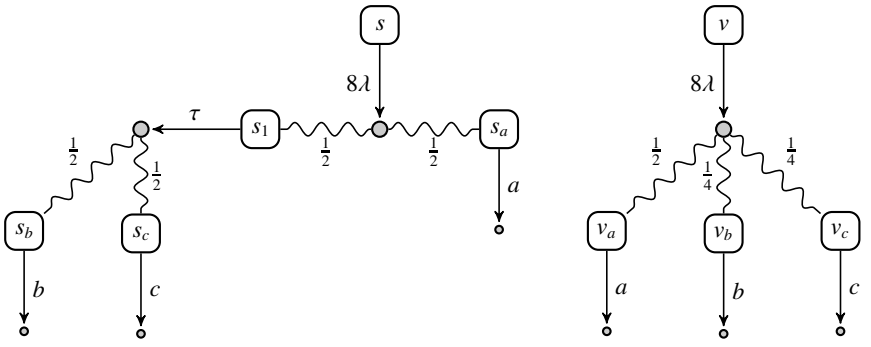


Fig. 3. Derived MLTSs of MAs in Figure 1

Definition 3. *Let $\mathcal{R} \subseteq S \times \mathcal{D}_{sub}(S)$ be a relation from states to subdistributions in an MLTS. Then $\overline{\mathcal{R}} \subseteq \mathcal{D}_{sub}(S) \times \mathcal{D}_{sub}(S)$ is the smallest relation that satisfies*

(i) $s\ \mathcal{R}\ \Theta$ implies $\overline{s}\ \overline{\mathcal{R}}\ \Theta$, and
(ii) $\Delta_i\ \overline{\mathcal{R}}\ \Theta_i$ for $i \in I$ implies $(\sum_{i\in I} p_i \cdot \Delta_i)\ \overline{\mathcal{R}}\ (\sum_{i\in I} p_i \cdot \Theta_i)$ for any $p_i \in [0,1]$ with $\sum_{i\in I} p_i \leq 1$. □

We apply this operation to the relations $\xrightarrow{\mu}$ in the MLTS for $\mu \in \mathsf{Act}_{\tau,\delta}$, where we also write $\xrightarrow{\mu}$ for $\overline{\xrightarrow{\mu}}$. Thus as source of a relation $\xrightarrow{\mu}$ we now also allow distributions, and even subdistributions.

Definition 4 (Hyper-derivations). *In an MLTS a hyper-derivation consists of a collection of subdistributions $\Delta, \Delta_k^{\rightarrow}, \Delta_k^{\times}$, for $k \geq 0$, with the following properties:*

$$\begin{aligned} \Delta &= \Delta_0^{\rightarrow} + \Delta_0^{\times} \\ \Delta_0^{\rightarrow} &\xrightarrow{\tau} \Delta_1^{\rightarrow} + \Delta_1^{\times} \\ &\vdots \\ \Delta_k^{\rightarrow} &\xrightarrow{\tau} \Delta_{k+1}^{\rightarrow} + \Delta_{k+1}^{\times} \\ &\vdots \\ \Delta' &= \sum_{k=0}^{\infty} \Delta_k^{\times} \end{aligned} \tag{1}$$

We call Δ' a hyper-derivative *of Δ, and write $\Delta \Longrightarrow \Delta'$.* □

We refer to [5] for more discussion about hyper-derivations.
With these concepts we can now define the appropriate notion of weak moves in a MLTS, with which we may then use to define our concept of bisimulations. We write $\Delta \overset{\tau}{\Longrightarrow} \Delta'$ to mean $\Delta \Longrightarrow \Delta'$ and $\Delta \overset{\alpha}{\Longrightarrow} \Delta'$, for $\alpha \in \mathsf{Act}_\delta \cup \mathbb{R}^+$, to mean $\Delta \Longrightarrow \xrightarrow{\lambda} \Longrightarrow \Delta'$.

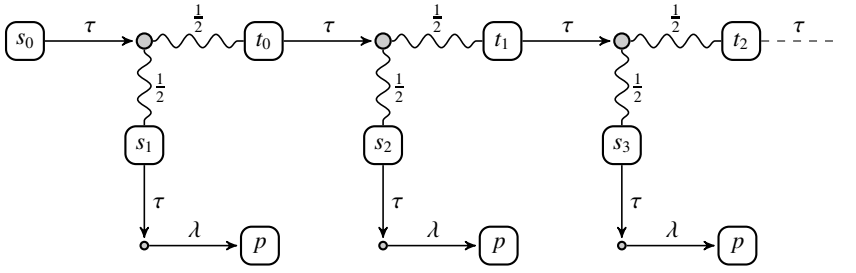


Fig. 4. Limiting internal moves

Definition 5 (Bisimulations). *For $\mathcal{R} \subseteq \mathcal{D}(S) \times \mathcal{D}(S)$, where S is the set of states in an MLTS, let $\mathcal{B}(\mathcal{R})$ be the relation over $\mathcal{D}(S) \times \mathcal{D}(S)$ determined by letting $\Delta\ \mathcal{B}(\mathcal{R})\ \Theta$ if, for each $\mu \in \mathsf{Act}_{\tau,\delta} \cup \mathbb{R}^+$ and all finite sets of probabilities $\{ p_i \mid i \in I \}$ satisfying $\sum_{i\in I} p_i = 1$,*

(i) whenever $\Delta \stackrel{\mu}{\Longrightarrow} \sum_{i\in I} p_i \cdot \Delta_i$, for any distributions Δ_i, there are distributions Θ_i with $\Theta \stackrel{\mu}{\Longrightarrow} \sum_{i\in I} p_i \cdot \Theta_i$, such that $\Delta_i\ \mathcal{R}\ \Theta_i$ for each $i \in I$

(ii) symmetrically, whenever $\Theta \stackrel{\mu}{\Longrightarrow} \sum_{i\in I} p_i \cdot \Theta_i$, for any distributions Θ_i, there are distributions Δ_i with $\Delta \stackrel{\mu}{\Longrightarrow} \sum_{i\in I} p_i \cdot \Delta_i$, such that $\Delta_i\ \mathcal{R}\ \Theta_i$ for each $i \in I$.

The largest solution to $\mathcal{R} = \mathcal{B}(\mathcal{R})$ is denoted by $\approx_{bis}$. □

Because of the form of the functional $\mathcal{B}$ it is easy to establish that $\approx_{bis}$ is an equivalence relation. However, due to the use of weak arrows, and the quantification over sets of probabilities, it is not easy to exhibit witness bisimulations. So we give an alternative characterisation of $\approx_{bis}$ in terms of a relation between *states* and distributions.

Definition 6 (Simple bisimulations). *For $\mathcal{R} \subseteq S \times \mathcal{D}(S)$, where again S is the set of states in an MLTS, let $\mathcal{SB}(\mathcal{R})$ be the relation over $S \times \mathcal{D}(S)$ defined by letting $s\ \mathcal{SB}(\mathcal{R})\ \Theta$ if, for each $\mu \in \mathsf{Act}_{\tau,\delta} \cup \mathbb{R}^+$,*

(i) whenever $s \stackrel{\mu}{\longrightarrow} \Delta'$, there is some $\Theta \stackrel{\mu}{\Longrightarrow} \Theta'$, such that $\Delta'\ \overline{\mathcal{R}}\ \Theta'$

(ii) there exists some $\Delta \in \mathcal{D}(S)$ such that $\overline{s} \stackrel{\tau}{\Longrightarrow} \Delta$ and $\Theta\ \overline{\mathcal{R}}\ \Delta$.

We use $\approx_{sbis}$ to denote the largest solution to $\mathcal{R} = \mathcal{SB}(\mathcal{R})$. □

Example 2. Consider again Figure 3. We have $s \approx_{sbis} \overline{v}$ because the following relation $\{\langle s, \overline{v}\rangle,\ \langle s_1, \frac{1}{2}\cdot\overline{v_b} + \frac{1}{2}\cdot\overline{v_c}\rangle,\ \langle s_a, \overline{v_a}\rangle,\ \langle s_b, \overline{v_b}\rangle,\ \langle s_c, \overline{v_c}\rangle,\ \langle v, \overline{s}\rangle,\ \langle v_a, \overline{s_a}\rangle,\ \langle v_b, \overline{s_b}\rangle,\ \langle v_c, \overline{s_c}\rangle\}$ is a simple bisimulation and therefore $s \approx_{sbis} \overline{v}$.

Consider the MA in Figure 4; starting from the initial state s_0 an ever increasing number of internal τ moves are performed before the eventual timed λ action, but with ever decreasing probability. The relation

$$\{\, \langle \lambda.p, \overline{s_i}\rangle,\ \langle \lambda.p, \overline{t_i}\rangle,\ \langle s_i, \overline{\lambda.p}\rangle,\ \langle t_i, \overline{\lambda.p}\rangle \mid i \geq 0 \,\}$$

is a simple bisimulation, and therefore $s_0 \approx_{sbis} \overline{\lambda.p}$, where $\lambda.p$ describes in an obvious manner the MA which does a timed action at rate λ and evolves to the state p.

Now consider the MA in Figure 2. We have $s \not\approx_{sbis} \overline{u}$ because the transition $s \xrightarrow{\tau} \frac{1}{2} \cdot \overline{s_1} + \frac{1}{2} \cdot \overline{s_2}$ cannot be matched by any transition from u. The state u cannot enable internal actions, so the only weak internal transition from $\overline{u}$ is $\overline{u} \stackrel{\tau}{\Longrightarrow} \overline{u}$. However, the derivative $\overline{u}$ is not able to simulate $\frac{1}{2} \cdot \overline{s_1} + \frac{1}{2} \cdot \overline{s_2}$ according to the lifted relation $\overline{\approx_{sbis}}$. Suppose for a contradiction that $(\frac{1}{2} \cdot \overline{s_1} + \frac{1}{2} \cdot \overline{s_2}) \; \overline{\approx_{sbis}} \; \overline{u}$. Then we must have $s_1 \approx_{sbis} \overline{u}$ and $s_2 \approx_{sbis} \overline{u}$; obviously neither of these hold. □

The two relations $\approx_{bis}$ and $\approx_{sbis}$ are closely related, as stated by the theorem below.

Theorem 1. *Let Δ and Θ be two distributions in a finitary MLTS.*

(i) If $\Delta \approx_{bis} \Theta$ then there is some Θ' with $\Theta \stackrel{\tau}{\Longrightarrow} \Theta'$ and $\Delta \; \overline{\approx_{sbis}} \; \Theta'$
(ii) If $\Delta \; \overline{\approx_{sbis}} \; \Theta$ then $\Delta \approx_{bis} \Theta$. □

For the remainder of the paper we will apply these relations, developed for MLTSs, to the states and distributions of MAs. For example we write $s \approx_{sbis} \Delta$, where s is a state in an MA M and Δ a distribution, to mean $s \approx_{sbis} \Delta$ in the derived $\mathsf{mlts}(M)$.

3 Composing Markov Automata

Here we assume that the set of actions Act is equipped with a complementation function $\bar{\cdot} : \mathsf{Act} \to \mathsf{Act}$ satisfying $\overline{\overline{a}} = a$; we say $\overline{a}$ is the complement of a. Then given two MAs, $M_i = \langle S_1, \mathsf{Act}_\tau, \to, \mapsto, \rangle$ for $i = 1, 2$, their composition $(M_1 \mid M_2)$ is given by $\langle S_1 \mid S_2, \mathsf{Act}_\tau, \to, \mapsto, \rangle$ where the set of states $S_1 \mid S_2 = \{ s_1 \mid s_2 \mid s_i \in S_i, i = 1, 2 \}$ and the relations are determined by the rules in Figure 5. The rules use the obvious extension of the function $\mid$ on pairs of states to pairs of distributions. To be precise $\Delta|\Theta$ is the distribution defined by: $(\Delta|\Theta)(s) = \Delta(s_1) \times \Theta(s_2)$ if $s = s_1|s_2$, and 0 otherwise. It can be checked that if M_1 and M_2 are Markov automata, then so is $(M_1 \mid M_2)$. We can internalise this composition relation by saying an Markov automaton M is *par-closed* if $(M \mid M)$ is already a sub-MA of M.

The simplest way of constructing a par-closed MA is by interpreting a process algebra as a universal Markov automaton. To this end we introduce the language mCCS whose terms are given by:

$$P, Q ::= \mathbf{0} \mid \delta.P \mid \lambda.D, \; \lambda \in \mathbb{R}^+ \mid \mu{:}D, \; \mu \in \mathsf{Act}_\tau \mid P + Q \mid P \mid Q \mid A$$
$$D ::= (\oplus_{i \in I} p_i \cdot P_i)$$

where A ranges over a set of process constants, with each of which is associated a definition, $A \Leftarrow \mathrm{Def}(A)$. mCCS is interpreted as an Markov automaton whose states are all the terms in the language, and whose arrows are determined by the rules in Figure 6, together with those in Figure 5; we have omitted the obvious symmetric counterparts to the rules (EXT.L), (EXT.L.L) and (EXT.D.L). Other operations, such as the standard hiding $Q\backslash a, a \in Act$, can also be easily given an interpretation. We say a process P from mCCS is *finitary* if the sub-MA consisting of all states reachable from P is finitary, and we use *finitary* mCCS to refer to the MA consisting of all such finitary P.

The rules (ACTION) and (DELAY) use the notation $[\![D]\!]$, where D has the form $(\oplus_{i \in I} p_i \cdot P_i)$, to denote the obvious distribution over process terms, whose support consists of

$$\frac{s \xrightarrow{\mu} \Delta}{s \mid t \xrightarrow{\mu} \Delta \mid \overline{t}}\ (\textsc{par.l}) \qquad \frac{t \xrightarrow{\mu} \Theta}{s \mid t \xrightarrow{\mu} \overline{s} \mid \Theta}\ (\textsc{par.r})$$

$$\frac{s \xrightarrow{a} \Delta,\ t \xrightarrow{\overline{a}} \Theta}{s \mid t \xrightarrow{\tau} \Delta \mid \Theta}\ (\textsc{par.i})$$

$$\frac{s \overset{\mathbf{d}}{\mapsto} \Delta,\ t \overset{\delta}{\mapsto} \Theta,\ s \mid t \not\xrightarrow{\tau}}{s \mid t \overset{\mathbf{d}}{\mapsto} \Delta \mid \Theta}\ (\textsc{par.l.t}) \qquad \frac{s \overset{\delta}{\mapsto} \Delta, t \overset{\mathbf{d}}{\mapsto} \Theta,\ s \mid t \not\xrightarrow{\tau}}{s \mid t \overset{\mathbf{d}}{\mapsto} \Delta \mid \Theta}\ \mathbf{d} = \delta, \lambda\ (\textsc{par.r.t})$$

Fig. 5. Composing Markov automata

$P_1, \ldots P_n$, each with weight p_i respectively. Most of the other rules should be self-explanatory, although the justification for the rules for λ transitions depends on non-trivial properties of exponential distributions, as explained in detail in [10].

Nevertheless, this interpretation of mCCS is quite different than that of other Markovian process calculi, such as those in [10,3]. First the actions $\mu : D$ are *insistent* rather than *lazy*; they do not allow time to pass. For example the process $(\lambda.Q \mid a:P)$ is stuck with respect to time; it can not perform any timed action. This is because the parallel operator requires each component to perform a timed action, which $a : P$ can not do, before time can pass. To obtain lazy actions one can define $a.P$ by the declaration $A \Leftarrow a:P + \delta.A$. Then we have the transition $\lambda.Q \mid a.P \overset{\lambda}{\mapsto} Q \mid a.P$ by an application of the rule (par.l.t) to the transitions $\lambda.Q \overset{\lambda}{\mapsto} Q$ and $a.P \overset{\delta}{\mapsto} a.P$.

The parallel operator is even more constraining in that at most one of its components can perform a definite delay. Again this is reminiscent of many existing Markovian process algebras [2,3], although these tend to have delays associated with external actions. But in the setting of mCCS the net effect is an operational semantics very similar to that in [8]. For example consider the process $Q = (\lambda_1.P_1 \mid \lambda_2.P_2)$. This has three timed actions:

(i) $Q \overset{\lambda_1}{\mapsto} (P_1 \mid \lambda_2.P_2)$ via an application of the rule (par.l.t) to the actions $\lambda_1.P_1 \overset{\lambda_1}{\mapsto} P_1$ and $\lambda_2.P_2 \overset{\delta}{\mapsto} \lambda_2.P_2$

(ii) $Q \overset{\lambda_2}{\mapsto} (\lambda_1 P_2 \mid P_2)$ via an application of (par.r.t) to the actions $\lambda_1.P_1 \overset{\delta}{\mapsto} \lambda_1.P$ and $\lambda_2.P_2 \overset{\lambda_2}{\mapsto} P_2$

(iii) $Q \overset{\delta}{\mapsto} Q$ via an application of either of (par.l.t) or (par.r.t) to the transitions $\lambda_1.P_1 \overset{\delta}{\mapsto} \lambda_1.P_1$ and $\lambda_1.P_1 \overset{\delta}{\mapsto} \lambda_1.P_1$.

Theorem 2 (Compositionality). *Let Δ, Θ and Γ be any distributions in a finitary par-closed MA. If $\Delta \approx_{bis} \Theta$ then $\Delta \mid \Gamma \approx_{bis} \Theta \mid \Gamma$.* □

$$\text{(ACTION)}\quad \mu{:}D \xrightarrow{\mu} [\![D]\!] \qquad \text{(RECURSION)}\quad \frac{\mathrm{Def}(A) \xrightarrow{\alpha} \Delta}{A \xrightarrow{\alpha} \Delta}\ \alpha = \mu, \lambda, \delta$$

$$\text{(EXT.L)}\quad \frac{P \xrightarrow{\mu} \Delta,}{P + Q \xrightarrow{\mu} \Delta} \qquad \text{(EXT.L.L)}\quad \frac{P \overset{\lambda}{\mapsto} \Delta,\ Q \not\xrightarrow{\tau}}{P + Q \overset{\lambda}{\mapsto} \Delta}$$

$$\text{(DELAY)}\quad \lambda.D \overset{\lambda}{\mapsto} [\![D]\!], \qquad \text{(D.}\delta\text{)}\quad \lambda.D \overset{\delta}{\mapsto} \overline{\lambda.D}$$

$$\text{(}\delta\text{.E)}\quad \frac{P \xrightarrow{\mu} \Delta}{\delta.P \xrightarrow{\mu} \Delta} \qquad \text{(}\delta\text{.D)}\quad \frac{P \not\xrightarrow{\tau}}{\delta.P \overset{\delta}{\mapsto} \overline{P}}$$

$$\text{(EXT)}\quad \frac{P \overset{\delta}{\mapsto} \Delta_1,\ Q \overset{\delta}{\mapsto} \Delta_2}{P + Q \overset{\delta}{\mapsto} \Delta_1 + \Delta_2} \qquad \text{(EXT.D.L)}\quad \frac{P \overset{\delta}{\mapsto} \Delta,\ Q \overset{\delta}{\not\mapsto},\ Q \not\xrightarrow{\tau}}{P + Q \overset{\delta}{\mapsto} \Delta}$$

Fig. 6. Operational semantics of mCCS

4 Soundness and Completeness

Consider an arbitrary par-closed MA $M = \langle S, \mathsf{Act}_\tau, \to, \mapsto \rangle$. Experimenting on processes in M consists in observing what communications a process can perform, as it evolves by both internal moves and the passage of time. To make this *evolution* precise let $\Delta \Longrightarrow \Delta'$ be the least reflexive relation satisfying:
(a) $\Delta \Longrightarrow \Delta_1$ and $\Delta_1 \xrightarrow{\tau} \Delta'$ implies $\Delta \Longrightarrow \Delta'$
(b) $\Delta \Longrightarrow \Delta_1$ and $\Delta_1 \xrightarrow{\lambda} \Delta'$ implies $\Delta \Longrightarrow \Delta'$, where $\lambda \in \mathbb{R}^+$
(c) $\Delta_i \Longrightarrow \Delta_i'$ for each $i \in I$ implies $(\sum_{i\in I} p_i \cdot \Delta_i) \Longrightarrow (\sum_{i\in I} p_i \cdot \Delta_i')$ where $\sum_{i\in I} p_i = 1$.

Thus $\Delta \Longrightarrow \Delta'$ is a relation between distributions in the $\mathsf{mlts}(M)$ which allows reduction either by internal actions τ or definite delay actions λ; with the latter the reductions are to distributions determined by the rates of the states in the support of Δ.

Definition 7 (Barbs). *For $\Delta \in \mathcal{D}(S)$ and $a \in \mathsf{Act}$, let $\mathcal{V}_a(\Delta) = \sum\{\Delta(s) \mid s \xrightarrow{a}\}$. We write $\Delta \Downarrow_a^{\geq p}$ whenever $\Delta \Longrightarrow \Delta'$, where $\mathcal{V}_a(\Delta') \geq p$.* □

Then we say a relation $\mathcal{R}$ is *barb-preserving* if $\Delta\ \mathcal{R}\ \Theta$ then $\Delta \Downarrow_a^{\geq p}$ iff $\Theta \Downarrow_a^{\geq p}$. It is *reduction-closed* if $\Delta\ \mathcal{R}\ \Theta$ implies
(i) whenever $\Delta \Longrightarrow \Delta'$, there is a $\Theta \Longrightarrow \Theta'$ such that $\Delta'\ \mathcal{R}\ \Theta'$
(ii) whenever $\Theta \Longrightarrow \Theta'$, there is a $\Delta \Longrightarrow \Delta'$ such that $\Delta'\ \mathcal{R}\ \Theta'$.

Finally, we say a relation $\mathcal{R}$ is *compositional* if $\Delta_1\ \mathcal{R}\ \Delta_2$ implies $(\Delta_1 \mid \Theta)\ \mathcal{R}\ (\Delta_2 \mid \Theta)$.

Definition 8. *In a par-closed MA, let $\approx_{rbc}$ be the largest relation over the states which is barb-preserving, reduction-closed and compositional.* □

Example 3. Consider the two processes $P_1 = \lambda_1.Q_1$ and $P_2 = \lambda_2.Q_2$ where $\lambda_1 < \lambda_2$ and Q_i are two arbitrary processes. We can show that $P_1 \not\approx_{rbc} P_2$ by exhibiting a *testing*

process T such that the barbs of $(P_1 \mid T)$ and $(P_2 \mid T)$ are different. For example let $T = \delta.\tau.\mathbf{0} + \lambda_1.succ$. In $(P_i \mid T)$ there is a race between two timed events; in $(P_2 \mid T)$ their rates are λ_1 versus λ_2 while in $(P_1 \mid T)$ both events have the same rate. If the timed event in the test wins out, the action *succ* will occur. Consequently $(P_1 \mid T) \Downarrow_{succ}^{\geq \frac{1}{2}}$. However $(P_2 \mid T)$ does not have this barb; instead $(P_2 \mid T) \Downarrow_{succ}^{\geq q}$, where $q = \frac{\lambda_1}{\lambda_1 + \lambda_2}$; q is strictly smaller than $\frac{1}{2}$ since $\lambda_1 < \lambda_2$.

It follows that $\lambda_1.\lambda_2.P \not\approx_{rbc} \lambda_2.\lambda_1.P$ when λ_1 and λ_2 are different. □

Example 4. Consider the processes $P_1 = a:Q$, $P_2 = a.Q$, and $P_3 = \lambda.P_2$, where Q is an arbitrary process, and we have seen that $a.Q$ is shorthand for a recursively defined process $A = a:Q + \delta.A$.

Note that according to our semantics P_1 does not let time pass. Let T be the testing process $\lambda.(\bar{a}.succ + \tau.\mathbf{0})$. The process $P_1 \mid T$ cannot evolve, thus $(P_1 \mid T) \not\Downarrow_{succ}^{>0}$[1]. However, we have $P_2 \mid T \overset{\lambda}{\mapsto} P_2 \mid (\bar{a}.succ + \tau.\mathbf{0}) \xrightarrow{\tau} Q \mid succ$, thus $(P_2 \mid T) \Downarrow_{succ}^{\geq 1}$. The only comparable barb for P_3 is $(P_3 \mid T) \Downarrow_{succ}^{\geq \frac{1}{2}}$, because if the timed event in the test takes place, then by maximal progress the τ action must happen before the timed event in the process. It follows that the three processes P_1, P_2 and P_3 can be distinguished. □

Although by definition $\approx_{rbc}$ is closed w.r.t. the evolution relation $\Longrightarrow$, in fact it is also closed w.r.t. the individual components, and indeed the definite delay operator.

Proposition 1. *Suppose $\Delta \approx_{rbc} \Theta$.*

(i) If $\Delta \overset{\mu}{\Longrightarrow} \Delta'$ with $\mu \in \mathsf{Act}_\tau$ then $\Theta \overset{\mu}{\Longrightarrow} \Theta'$ such that $\Delta' \approx_{rbc} \Theta'$.
(ii) If $\Delta \overset{\lambda}{\Longrightarrow} \Delta'$ with $\lambda \in \mathbb{R}^+$ then $\Theta \overset{\lambda}{\Longrightarrow} \Theta'$ such that $\Delta' \approx_{rbc} \Theta'$.
(iii) If $\Delta \overset{\delta}{\Longrightarrow} \Delta'$ then $\Theta \overset{\delta}{\Longrightarrow} \Theta'$ such that $\Delta' \approx_{rbc} \Theta'$. □

Example 5. Consider the two MAs s and u from Figure 2, discussed in the Introduction. Suppose $s \approx_{rbc} u$. Then by compositionality we have $s \mid T \approx_{rbc} u \mid T$, where T is the process $\tau.\delta.\bar{a}.succ + \tau.\delta.\bar{b}.succ$. Let Δ denote the point distribution $\overline{\mathbf{0} \mid succ}$. Since $s \mid T \Longrightarrow \Delta$, we have $(s \mid T) \Downarrow_{succ}^{\geq 1}$.

However, the weak derivatives of $u \mid T$ under the evolution relation are very few, and one can easily check that none will have exactly the barbs of Δ because if $(u \mid T) \Downarrow_{succ}^{\geq p}$ then p is at most $\frac{1}{2}$. It follows that s and u are indeed behaviourally different. □

Theorem 3 (Soundness). *In a finitary par-closed MA, if $\Delta \approx_{bis} \Theta$ then $\Delta \approx_{rbc} \Theta$.* □

In order to obtain *completeness*, the converse of Theorem 3, we need to ensure that the MA under consideration can provide sufficient contexts in order to probe the behaviour of systems. For this purpose, we use the language mCCS.

Theorem 4 (Completeness). *In finitary mCCS, $\Delta \approx_{rbc} \Theta$ implies $\Delta \approx_{bis} \Theta$.* □

5 Conclusion and Related Work

The thesis underlying this paper is that bisimulations should be considered as a proof methodology for demonstrating behavioural equivalence between systems, rather than

[1] Here we use the notation $P \not\Downarrow_a^{>0}$ to mean that $P \Downarrow_a^{\geq p}$ does not hold for any $p > 0$.

providing the definition of the extensional behavioural equivalence itself. We have adapted the well-known *reduction barbed congruence* used for a variety of process calculi [13,19,9], to obtain a touchstone extensional behavioural equivalence for a minor variation of the Markov automata, MAs, originally defined in [8]. Incidently there are also minor variations on the formulation of reduction barbed congruence, often called *contextual equivalence* or *barbed congruence*, in the literature. See [9,21] for a discussion of the differences.

Then we have defined a novel notion of (weak) bisimulations which provide both a sound and complete coinductive proof methodology for establishing the equivalence between such automata. These results were achieved within the context of a rich language, mCCS, for defining MAs. Of particular significance is the presence of insistent actions and a compositional operator which is sensitive to the passage of time; this combination is reminiscent of synchronous CCS [16], although similar compositional operators have already been used for certain varieties of Markov processes [3]. We should point out that our interpretation of mCCS is somewhat simplistic, in that unlike IMC in [10] it does not take into account the multiplicities of action occurrences. However, our interpretation is sufficient for the purposes of this paper. If we were interested in, for example, developing an algebraic theory for mCCS then a more refined interpretation would be required; this could easily be adapted from [10].

There are already quite a few variations on the theme of bisimulations for PAs which can be used to establish behavioural equivalences between MAs [24,18,15,6,11]. A characteristic of our formulation is that it allows bisimulations to relate states to distributions rather than simply states, thus differentiating it from most of these. One exception is [8], where properties of subdistributions are also used in defining their bisimulations. However, our bisimulation $\approx_{bis}$ is different from the bisimulation of [8], denoted by $\approx_{MA}$ here, because the former is defined for full distributions while the latter is for subdistributions. Even if we restrict $\approx_{MA}$ to full distributions, they are still different. For example, we have $A \approx_{bis} \mathbf{0}$ but $A \not\approx_{MA} \mathbf{0}$, where $A \Leftarrow \tau{:}A$. We conjecture that in general $\approx_{bis}$ is strictly coarser than $\approx_{MA}$ (restricted to full distributions), but they coincide for non-divergent systems [7].

Our approach to Markov processes is based directly on that of [8,10], in which external actions are considered instantaneous, and time can only pass when no more internal activity can be performed. Moreover it is only timed actions which are subject to Markovian behaviour. However, there is a large literature on a more general framework in which Markovian behaviour applies to all actions. See [12] or Chapter 3 of [1] for a representative exposition. It would be interesting to see if our notion of bisimulation could be adapted to such a framework.

Acknowledgement. We thank Christian Eisentraut for the interesting discussion on clarifying the relationship between $\approx_{bis}$ and $\approx_{MA}$.

References

1. Aldini, A., Bernardo, M., Corradini, F.: A Process Algebraic Approach to Software Architecture Design. Springer, Heidelberg (2010)
2. Bernardo, M., Gorrieri, R.: A tutorial on empa: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. Theor. Comput. Sci. 202(1-2), 1–54 (1998)

3. Bravetti, M., Bernardo, M.: Compositional asymmetric cooperations for process algebras with probabilities, priorities, and time. ENTCS 39(3) (2000)
4. Deng, Y., Du, W.: Probabilistic barbed congruence. ENTCS 190(3), 185–203 (2007)
5. Deng, Y., van Glabbeek, R., Hennessy, M., Morgan, C.: Testing finitary probabilistic processes (extended abstract). In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 274–288. Springer, Heidelberg (2009)
6. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Weak bisimulation is sound and complete for pCTL*. Information and Computation 208(2), 203–219 (2010)
7. Eisentraut, C.: Personal communication (2011)
8. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: Proc. LICS 2010, pp. 342–351 (2010)
9. Fournet, C., Gonthier, G.: A hierarchy of equivalences for asynchronous calculi. J. Log. Algebr. Program. 63(1), 131–173 (2005)
10. Hermanns, H.: Interactive Markov Chains: The Quest for Quantified Quality. LNCS, vol. 2428, p. 129. Springer, Heidelberg (2002)
11. Hermanns, H., Parma, A., Segala, R., Wachter, B., Zhang, L.: Probabilistic logical characterization. Inf. Comput. 209, 154–172 (2011)
12. Hillston, J.: A Compositional Approach to Performance Modelling. Distinguished Dissertations in Computer Science. Cambridge University Press, New York (2005)
13. Honda, K., Tokoro, M.: On asynchronous communication semantics. In: Zatarain-Cabada, R., Wang, J. (eds.) ECOOP-WS 1991. LNCS, vol. 612, pp. 21–51. Springer, Heidelberg (1992)
14. Jeffrey, A., Rathke, J.: Contextual equivalence for higher-order pi-calculus revisited. LMCS 1(1:4) (2005)
15. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing (preliminary report). In: Proc. POPL 1989, pp. 344–352. ACM, New York (1989)
16. Milner, R.: Calculi for synchrony and asynchrony. Theor. Comput. Sci. 25, 267–310 (1983)
17. Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)
18. Philippou, A., Lee, I., Sokolsky, O.: Weak bisimulation for probabilistic systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 334–349. Springer, Heidelberg (2000)
19. Rathke, J., Sobocinski, P.: Deriving structural labelled transitions for mobile ambients. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 462–476. Springer, Heidelberg (2008)
20. Sangiorgi, D., Kobayashi, N., Sumii, E.: Environmental bisimulations for higher-order languages. In: Proc. LICS 2007, pp. 293–302. IEEE Computer Society, Los Alamitos (2007)
21. Sangiorgi, D., Walker, D.: The π-calculus: a Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)
22. Segala, R.: Modeling and verification of randomized distributed real-time systems. Technical Report MIT/LCS/TR-676, PhD thesis, MIT, Dept. of EECS (1995)
23. Segala, R.: Testing probabilistic automata. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 299–314. Springer, Heidelberg (1996)
24. Segala, R., Lynch, N.A.: Probabilistic simulations for probabilistic processes. Nord. J. Comput. 2(2), 250–273 (1995)

Runtime Analysis of Probabilistic Programs with Unbounded Recursion

Tomáš Brázdil[1,⋆], Stefan Kiefer[2,⋆⋆],
Antonín Kučera[1,⋆], and Ivana Hutařová Vařeková[1,⋆⋆⋆]

[1] Faculty of Informatics, Masaryk University, Czech Republic
{brazdil,kucera}@fi.muni.cz, ivarekova@centrum.cz
[2] Department of Computer Science, University of Oxford, United Kingdom
stefan.kiefer@cs.ox.ac.uk

Abstract. We study the runtime in probabilistic programs with unbounded recursion. As underlying formal model for such programs we use *probabilistic pushdown automata (pPDA)* which exactly correspond to recursive Markov chains. We show that every pPDA can be transformed into a stateless pPDA (called "pBPA") whose runtime and further properties are closely related to those of the original pPDA. This result substantially simplifies the analysis of runtime and other pPDA properties. We prove that for every pPDA the probability of performing a long run decreases *exponentially* in the length of the run, if and only if the expected runtime in the pPDA is *finite*. If the expectation is infinite, then the probability decreases "polynomially". We show that these bounds are asymptotically tight. Our tail bounds on the runtime are *generic*, i.e., applicable to any probabilistic program with unbounded recursion. An intuitive interpretation is that in pPDA the runtime is exponentially unlikely to deviate from its expected value.

1 Introduction

We study the termination time in programs with unbounded recursion, which are either randomized or operate on statistically quantified inputs. As underlying formal model for such programs we use *probabilistic pushdown automata (pPDA)* [13,14,6,3] which are equivalent to recursive Markov chains [18,16,17]. Since pushdown automata are a standard and well-established model for programs with recursive procedure calls, our abstract results imply *generic* and *tight* tail bounds for termination time, the main performance characteristic of probabilistic recursive programs.

A pPDA consists of a finite set of *control states*, a finite *stack alphabet*, and a finite set of *rules* of the form $pX \overset{x}{\hookrightarrow} q\alpha$, where p, q are control states, X is a stack symbol, α is a finite sequence of stack symbols (possibly empty), and $x \in (0, 1]$ is the (rational)

⋆ Tomáš Brázdil and Antonín Kučera are supported by the Institute for Theoretical Computer Science (ITI), project No. 1M0545, and by the Czech Science Foundation, grant No. P202/10/1469.

⋆⋆ Stefan Kiefer is supported by a postdoctoral fellowship of the German Academic Exchange Service (DAAD).

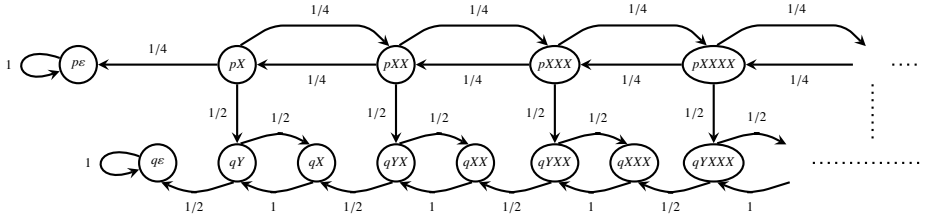
⋆⋆⋆ Ivana Hutařová Vařeková is supported by by the Czech Science Foundation, grant No. 102/09/H042.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 319–331, 2011.

probability of the rule. We require that for each pX, the sum of the probabilities of all rules of the form $pX \overset{x}{\hookrightarrow} q\alpha$ is equal to 1. Each pPDA Δ induces an infinite-state Markov chain M_Δ, where the states are configurations of the form $p\alpha$ (p is the current control state and α is the current stack content), and $pX\beta \xrightarrow{x} q\alpha\beta$ is a transition of M_Δ iff $pX \overset{x}{\hookrightarrow} q\alpha$ is a rule of Δ. We also stipulate that $p\varepsilon \xrightarrow{1} p\varepsilon$ for every control state p, where ε denotes the empty stack. For example, consider the pPDA $\hat{\Delta}$ with two control states p, q, two stack symbols X, Y, and the rules

$$pX \overset{1/4}{\hookrightarrow} p\varepsilon,\ pX \overset{1/4}{\hookrightarrow} pXX,\ pX \overset{1/2}{\hookrightarrow} qY,\ pY \overset{1}{\hookrightarrow} pY,\ qY \overset{1/2}{\hookrightarrow} qX,\ qY \overset{1/2}{\hookrightarrow} q\varepsilon,\ qX \overset{1}{\hookrightarrow} qY\,.$$

The structure of Markov chain $M_{\hat{\Delta}}$ is indicated below.



pPDA can model programs that use unbounded "stack-like" data structures such as stacks, counters, or even queues (in some cases, the exact ordering of items stored in a queue is irrelevant and the queue can be safely replaced with a stack). Transition probabilities may reflect the random choices of the program (such as "coin flips" in randomized algorithms) or some statistical assumptions about the input data. In particular, pPDA model *recursive* programs. The global data of such a program are stored in the finite control, and the individual procedures and functions together with their local data correspond to the stack symbols (a function call/return is modeled by pushing/popping the associated stack symbol onto/from the stack). As a simple example, consider the recursive program *Tree* of Figure 1, which computes the value of an *And/Or-tree*, i.e., a tree such that (i) every node has either zero or two children, (ii) every inner node is either an And-node or an Or-node, and (iii) on any path from the root to a leaf And- and Or-nodes alternate. We further assume that the root is either a leaf or an And-node. *Tree* starts by invoking the function `And` on the root of a given And/Or-tree. Observe that the program evaluates subtrees only if necessary. Now assume that the input are random And/Or trees following the Galton-Watson distribution: a node of the tree has two children with probability $1/2$, and no children with probability $1/2$. Furthermore, the conditional probabilities that a childless node evaluates to 0 and 1 are also both equal to $1/2$. On inputs with this distribution, the algorithm corresponds to a pPDA Δ_{Tree} of Figure 1 (the control states r_0 and r_1 model the return values 0 and 1).

We study the *termination time* of runs in a given pPDA Δ. For every pair of control states p, q and every stack symbol X of Δ, let $Run(pXq)$ be the set of all runs (infinite paths) in M_Δ initiated in pX which visit $q\varepsilon$. The termination time is modeled by the random variable $\mathbf{T}_{pX}$, which to every run w assigns either the number of steps needed to reach a configuration with empty stack, or ∞ if there is no such configuration. The conditional expected value $\mathbb{E}[\mathbf{T}_{pX} \mid Run(pXq)]$, denoted just by $E[pXq]$ for short, then corresponds to the average number of steps needed to reach $q\varepsilon$ from pX, computed only for those runs initiated in pX which terminate in $q\varepsilon$. For example, using the results of

```
function And(node)
  if node.leaf then
    return node.value
  else
    v := Or(node.left)
    if v = 0 then
      return 0
    else
      return Or(node.right)
```

```
function Or(node)
  if node.leaf then
    return node.value
  else
    v := And(node.left)
    if v = 1 then
      return 1
    else
      return And(node.right)
```

$$qA \xhookrightarrow{1/4} r_1\varepsilon \quad qO \xhookrightarrow{1/4} r_1\varepsilon$$
$$qA \xhookrightarrow{1/4} r_0\varepsilon \quad qO \xhookrightarrow{1/4} r_0\varepsilon$$
$$qA \xhookrightarrow{1/2} qOA \quad qO \xhookrightarrow{1/2} qAO$$
$$r_0A \xhookrightarrow{1} r_0\varepsilon \quad r_1O \xhookrightarrow{1} r_1\varepsilon$$
$$r_1A \xhookrightarrow{1} qO \quad r_0O \xhookrightarrow{1} qA$$

Fig. 1. The program *Tree* and its pPDA model Δ_{Tree}

[13,14,18], one can show that the functions And and Or of the program *Tree* terminate with probability one, and the expected termination times can be computed by solving a system of linear equations. Thus, we obtain the following:

$$E[qAr_0] = 7.155113 \quad E[qOr_0] = 7.172218 \quad E[r_0Ar_0] = 1.000000 \quad E[r_1Or_1] = 1.000000$$
$$E[qAr_1] = 7.172218 \quad E[qOr_1] = 7.155113 \quad E[r_1Ar_0] = 8.172218 \quad E[r_0Or_1] = 8.172218$$
$$E[r_1Ar_1] = 8.155113 \quad E[r_0Or_0] = 8.155113$$

However, the mere expectation of the termination time does not provide much information about its distribution until we analyze the associated *tail bound*, i.e., the probability that the termination time deviates from its expected value by a given amount. That is, we are interested in bounds for the conditional probability $\mathcal{P}(\mathbf{T}_{pX} \geq n \mid Run(pXq))$. (Note this probability makes sense regardless of whether $E[pXq]$ is finite or infinite.) Assuming that the (conditional) expectation and variance of $\mathbf{T}_{pX}$ are finite, one can apply Markov's and Chebyshev's inequalities and thus yield bounds of the form $\mathcal{P}(\mathbf{T}_{pX} \geq n \mid Run(pXq)) \leq c/n$ and $\mathcal{P}(\mathbf{T}_{pX} \geq n \mid Run(pXq)) \leq c/n^2$, respectively, where c is a constant depending only on the underlying pPDA. However, these bounds are asymptotically always worse than our exponential bound (see below). If $E[pXq]$ is infinite, these inequalities cannot be used at all.

Our contribution. The main contributions of this paper are the following:

- We show that every pPDA can be effectively transformed into a *stateless* pPDA (called "pBPA") so that all important quantitative characteristics of runs are preserved. This simple (but fundamental) observation was overlooked in previous works on pPDA and related models [13,14,6,3,18,16,17], although it simplifies virtually all of these results. Hence, we can w.l.o.g. concentrate just on the study of pBPA. Moreover, for the runtime analysis, the transformation yields a pBPA all of whose symbols terminate with probability one, which further simplifies the analysis.
- We provide tail bounds for $\mathbf{T}_{pX}$ which are *asymptotically optimal for every pPDA* and are applicable also in the case when $E[pXq]$ is infinite. More precisely, we show that for every pair of control states p, q and every stack symbol X, there are essentially three possibilities:
  - There is a "small" k such that $\mathcal{P}(\mathbf{T}_{pX} \geq n \mid Run(pXq)) = 0$ for all $n \geq k$.
  - $E[pXq]$ is finite and $\mathcal{P}(\mathbf{T}_{pX} \geq n \mid Run(pXq))$ decreases exponentially in n.
  - $E[pXq]$ is infinite and $\mathcal{P}(\mathbf{T}_{pX} \geq n \mid Run(pXq))$ decreases "polynomially" in n.

The exact formulation of this result, including the explanation of what is meant by a "polynomial" decrease, is given in Theorem 7 (technically, Theorem 7 is formulated for pBPA which terminate with probability one, which is no restriction as explained above). Observe that a direct consequence of the above theorem is that *all* conditional moments $\mathbb{E}[\mathbf{T}_{pX}^k \mid Run(pXq)]$ are simultaneously either finite or infinite (in particular, if $E[pXq]$ is finite, then so is the conditional variance of $\mathbf{T}_{pX}$).

The characterization given in Theorem 7 is effective. In particular, it is decidable in polynomial space whether $E[pXq]$ is finite or infinite by using the results of [13,14,18], and if $E[pXq]$ is finite, we can compute concrete bounds on the probabilities. Our results vastly improve on what was previously known on the termination time $\mathbf{T}_{pX}$. Previous work, in particular [14,2], has focused on computing expectations and variances for a class of random variables on pPDA runs, a class that includes $\mathbf{T}_{pX}$ as prime example. Note that our exponential bound given in Theorem 7 depends, like Markov's inequality, only on expectations, which can be efficiently approximated by the methods of [14,12].

An intuitive interpretation of our results is that pPDA with finite (conditional) expected termination time are well-behaved in the sense that the termination time is exponentially unlikely to deviate from its expectation. Of course, a detailed analysis of a concrete pPDA may lead to better bounds, but these bounds will be *asymptotically equivalent* to our generic bounds. Also note that the conditional expected termination time can be finite even for pPDA that do not terminate with probability one. Hence, for every $\varepsilon > 0$ we can compute a tight threshold k such that if a given pPDA terminates at all, it terminates after at most k steps with probability $1-\varepsilon$ (this is useful for interrupting programs that are supposed but not guaranteed to terminate).

Proof techniques. The main mathematical tool for establishing our results on runtime is (basic) martingale theory and its tools such as the optional stopping theorem and Azuma's inequality (see Section 3.2). More precisely, we construct two different martingales corresponding to the cases when the expected termination time is finite resp. infinite. In combination with our reduction to pBPA this establishes a powerful link between pBPA, pPDA, and martingale theory.

Our analysis of termination time in the case when the expected termination time is infinite builds on Perron-Frobenius theory for nonnegative matrices as well as on recent results from [18,12]. We also use some of the observations presented in [13,14,6].

Related work. The application of Azuma's inequality in the analysis of particular randomized algorithms is also known as the *method of bounded differences*; see, e.g., [24,10] and the references therein. In contrast, we apply martingale methods not to particular algorithms, but to the pPDA model as a whole.

Analyzing the distribution of termination time is closely related to the analysis of multitype branching processes (MT-BPs) [19]. A MT-BP is very much like a pBPA (see above). The stack symbols in pBPA correspond to species in MT-BPs. An ε-rule corresponds to the death of an individual, whereas a rule with two or more symbols on the right hand side corresponds to reproduction. Since in MT-BPs the symbols on the right hand side of rules evolve concurrently, termination time in pBPA does *not* correspond to extinction time in MT-BPs, but to the size of the *total progeny* of an individual, i.e., the number of direct or indirect descendants of an individual. The distribution of the total progeny of a MT-BP has been studied mainly for the case of a single species, see,

e.g., [19,25,26] and the references therein, but to the best of our knowledge, no tail bounds for MT-BPs have been given. Hence, Theorem 7 can also be seen as a contribution to MT-BP theory.

Stochastic context-free grammars (SCFGs) [23] are also closely related to pBPA. The termination time in pBPA corresponds to the number of nodes in a derivation tree of a SCFG, so our analysis of pBPA immediately applies to SCFGs. Quasi-Birth-Death processes (QBDs) can also be seen as a special case of pPDA. A QBD is a generalization of a birth-death process studied in queueing theory and applied probability (see, e.g., [22,1,15]). Intuitively, a QBD describes an unbounded queue, using a counter to count the number of jobs in the queue, where the queue can be in one of finitely many distinct "modes". Hence, a (discrete-time) QBD can be equivalently defined by a pPDA with one stack symbol used to emulate the counter. These special pPDA are also known as *probabilistic one-counter automata (pOC)* [15,5,4]. Recently, it has been shown in [7] that every pOC induces a martingale apt for studying the properties of both terminating and nonterminating runs in pOC. The construction is based on ideas specific to pOC that are completely unrelated to the ones presented in this paper.

Previous work on pPDA and the equivalent model of recursive Markov chains includes [13,14,6,3,18,16,17]. In this paper we use many of the results presented in these papers, which is explicitly acknowledged at appropriate places. Missing proofs can be found in [8].

2 Preliminaries

In the rest of this paper, $\mathbb{N}$, $\mathbb{N}_0$, and $\mathbb{R}$ denote the set of positive integers, non-negative integers, and real numbers, respectively. The tuples of $A_1 \times A_2 \cdots \times A_n$ are often written simply as $a_1 a_2 \ldots a_n$. The set of all finite words over a given alphabet Σ is denoted by Σ^*, and the set of all infinite words over Σ is denoted by Σ^ω. We write ε for the empty word. The length of a given $w \in \Sigma^* \cup \Sigma^\omega$ is denoted by $|w|$, where the length of an infinite word is ∞. Given a word (finite or infinite) over Σ, the individual letters of w are denoted by $w(0), w(1), \ldots$

Definition 1 (Markov Chains). *A* Markov chain *is a triple $M = (S, \to, Prob)$ where S is a finite or countably infinite set of* states*, $\to \subseteq S \times S$ is a* transition relation*, and $Prob$ is a function which to each transition $s \to t$ of M assigns its probability $Prob(s \to t) > 0$ so that for every $s \in S$ we have $\sum_{s \to t} Prob(s \to t) = 1$ (as usual, we write $s \xrightarrow{x} t$ instead of $Prob(s \to t) = x$).*

A *path* in M is a finite or infinite word $w \in S^+ \cup S^\omega$ such that $w(i-1) \to w(i)$ for every $1 \leq i < |w|$. A *run* in M is an infinite path in M. We denote by $Run[M]$ the set of all runs in M. The set of all runs that start with a given finite path w is denoted by $Run[M](w)$. When M is understood, we write just Run and $Run(w)$ instead of $Run[M]$ and $Run[M](w)$, respectively. Given $s \in S$ and $A \subseteq S$, we say A *is reachable from* s if there is a run w such that $w(0) = s$ and $w(i) \in A$ for some $i \geq 0$.

To every $s \in S$ we associate the probability space $(Run(s), \mathcal{F}, \mathcal{P})$ where $\mathcal{F}$ is the σ-field generated by all *basic cylinders* $Run(w)$ where w is a finite path starting with s, and $\mathcal{P} : \mathcal{F} \to [0,1]$ is the unique probability measure such that $\mathcal{P}(Run(w)) = \Pi_{i=1}^{|w|-1} x_i$

where $w(i-1) \xrightarrow{x_i} w(i)$ for every $1 \leq i < |w|$. If $|w| = 1$, we put $\mathcal{P}(Run(w)) = 1$. Note that only certain subsets of $Run(s)$ are $\mathcal{P}$-measurable, but in this paper we only deal with "safe" subsets that are guaranteed to be in $\mathcal{F}$.

Definition 2 (probabilistic PDA). *A* probabilistic pushdown automaton (pPDA) *is a tuple* $\Delta = (Q, \Gamma, \hookrightarrow, Prob)$ *where* Q *is a finite set of* control states, Γ *is a finite* stack alphabet, $\hookrightarrow \subseteq (Q \times \Gamma) \times (Q \times \Gamma^{\leq 2})$ *is a* transition relation *(where* $\Gamma^{\leq 2} = \{\alpha \in \Gamma^* , |\alpha| \leq 2\}$*), and Prob is a function which to each transition* $pX \hookrightarrow q\alpha$ *assigns its probability* $Prob(pX \hookrightarrow q\alpha) > 0$ *so that for all* $p \in Q$ *and* $X \in \Gamma$ *we have that* $\sum_{pX \hookrightarrow q\alpha} Prob(pX \hookrightarrow q\alpha) = 1$. *As usual, we write* $pX \overset{x}{\hookrightarrow} q\alpha$ *instead of* $Prob(pX \hookrightarrow q\alpha) = x$.

Elements of $Q \times \Gamma^*$ are called *configurations* of Δ. A pPDA with just one control state is called pBPA.[1] In what follows, configurations of pBPA are usually written without the (only) control state p (i.e., we write just α instead of $p\alpha$). We define the *size* of a pPDA Δ as $|\Delta| = |Q| + |\Gamma| + |\hookrightarrow| + |Prob|$, where $|Prob|$ is the sum of sizes of binary representations of values taken by $Prob$. To Δ we associate the Markov chain M_Δ with $Q \times \Gamma^*$ as the set of states and transitions defined as follows:

- $p\varepsilon \xrightarrow{1} p\varepsilon$ for each $p \in Q$;
- $pX\beta \xrightarrow{x} q\alpha\beta$ is a transition of M_Δ iff $pX \overset{x}{\hookrightarrow} q\alpha$ is a transition of Δ.

For all $pXq \in Q \times \Gamma \times Q$ and $rY \in Q \times \Gamma$, we define

- $Run(pXq) = \{w \in Run(pX) \mid w(i) = q\varepsilon \text{ for some } i \in \mathbb{N}\}$
- $Run(rY\uparrow) = Run(rY) \setminus \bigcup_{s \in Q} Run(rYs)$.

Further, we put $[pXq] = \mathcal{P}(Run(pXq))$ and $[pX\uparrow] = \mathcal{P}(Run(pX\uparrow))$. If Δ is a pBPA, we write $[X]$ and $[X\uparrow]$ instead of $[pXp]$ and $[pX\uparrow]$, where p is the only control state of Δ.

Let $p\alpha \in Q \times \Gamma^*$. We denote by $\mathbf{T}_{p\alpha}$ a random variable over $Run(p\alpha)$ where $\mathbf{T}_{p\alpha}(w)$ is either the least $n \in \mathbb{N}_0$ such that $w(n) = q\varepsilon$ for some $q \in Q$, or ∞ if there is no such n. Intuitively, $\mathbf{T}_{p\alpha}(w)$ is the number of steps ("the time") in which the run w initiated in $p\alpha$ terminates.

3 The Results

In this section we present the results outlined in Section 1. More precisely, in Section 3.1 we show how to transform a given pPDA into an equivalent pBPA, and in Section 3.2 we design the promised martingales and derive our tight tail bounds for the termination probability.

3.1 Transforming pPDA into pBPA

Let $\Delta = (Q, \Gamma, \hookrightarrow, Prob)$ be a pPDA. We show how to construct a pBPA $\Delta_\bullet$ which is "equivalent" to Δ in a well-defined sense. This construction is a relatively straightforward modification of the standard method for transforming a PDA into an equivalent

[1] The "BPA" acronym stands for "Basic Process Algebra" and it is used mainly for historical reasons. pBPA are closely related to stochastic context-free grammars and are also called *1-exit recursive Markov chains* (see, e.g., [18]).

context-free grammar (see, e.g., [20]), but has so far been overlooked in the existing literature on probabilistic PDA. The idea behind this method is to construct a BPA with stack symbols of the form $\langle pXq\rangle$ for all $p, q \in Q$ and $X \in \Gamma$. Roughly speaking, such a triple corresponds to terminating paths from pX to $q\varepsilon$. Subsequently, transitions of the BPA are induced by transitions of the PDA in a way corresponding to this intuition. For example, a transition of the form $pX \hookrightarrow rYZ$ induces transitions of the form $\langle pXq\rangle \hookrightarrow \langle rYs\rangle\langle sZq\rangle$ for all $s \in Q$. Then each path from pX to $q\varepsilon$ maps naturally to a path from $\langle pXq\rangle$ to ε. This construction can also be applied in the probabilistic setting by assigning probabilities to transitions so that the probability of the corresponding paths is preserved. We also deal with nonterminating runs by introducing new stack symbols of the form $\langle pX\uparrow\rangle$.

Formally, the stack alphabet of $\Delta_\bullet$ is defined as follows: For every $pX \in Q \times \Gamma$ such that $[pX\uparrow] > 0$ we add a stack symbol $\langle pX\uparrow\rangle$, and for every $pXq \in Q \times \Gamma \times Q$ such that $[pXq] > 0$ we add a stack symbol $\langle pXq\rangle$. Note that the stack alphabet of $\Delta_\bullet$ is effectively constructible in polynomial space by applying the results of [13,18].

Now we construct the rules $\hookrightarrow_\bullet$ of $\Delta_\bullet$. For all $\langle pXq\rangle$ we have the following rules:

- if $pX \overset{x}{\hookrightarrow} rYZ$ in Δ, then for all $s \in Q$ such that $y = x \cdot [rYs] \cdot [sZq] > 0$ we put $\langle pXq\rangle \xhookrightarrow{y/[pXq]}_\bullet \langle rYs\rangle\langle sZq\rangle$;
- if $pX \overset{x}{\hookrightarrow} rY$ in Δ, where $y = x \cdot [rYq] > 0$, we put $\langle pXq\rangle \xhookrightarrow{y/[pXq]}_\bullet \langle rYq\rangle$;
- if $pX \overset{x}{\hookrightarrow} q\varepsilon$ in Δ, we put $\langle pXq\rangle \xhookrightarrow{x/[pXq]}_\bullet \varepsilon$.

For all $\langle pX\uparrow\rangle$ we have the following rules:

- if $pX \overset{x}{\hookrightarrow} rYZ$ in Δ, then for every $s \in Q$ where $y = x \cdot [rYs] \cdot [sZ\uparrow] > 0$ we add $\langle pX\uparrow\rangle \xhookrightarrow{y/[pX\uparrow]}_\bullet \langle rYs\rangle\langle sZ\uparrow\rangle$;
- for all $qY \in Q \times \Gamma$ where $x = [qY\uparrow] \cdot \sum_{pX \hookrightarrow qY\beta} Prob(pX \hookrightarrow qY\beta) > 0$, we add $\langle pX\uparrow\rangle \xhookrightarrow{x/[pX\uparrow]}_\bullet \langle qY\uparrow\rangle$.

Note that the transition probabilities of $\Delta_\bullet$ may take irrational values. Still, the construction of $\Delta_\bullet$ is to some extent "effective" due to the following proposition:

Proposition 3 ([13,18]). *Let $\Delta = (Q, \Gamma, \hookrightarrow, Prob)$ be a pPDA. Let $pXq \in Q \times \Gamma \times Q$. There is a formula $\Phi(x)$ of $ExTh(\mathbb{R})$ (the existential theory of the reals) with one free variable x such that the length of $\Phi(x)$ is polynomial in $|\Delta|$ and $\Phi(x/r)$ is valid iff $r = [pXq]$.*

Using Proposition 3, one can compute formulae of $ExTh(\mathbb{R})$ that "encode" transition probabilities of $\Delta_\bullet$. Moreover, these probabilities can be effectively approximated up to an arbitrarily small error by employing either the decision procedure for $ExTh(\mathbb{R})$ [9] or by using Newton's method [11,21,12].

Example 4. Consider a pPDA Δ with two control states, p, q, one stack symbol, X, and the following transition rules:

$$pX \overset{a}{\hookrightarrow} qXX,\ pX \overset{1-a}{\hookrightarrow} q\varepsilon,\ qX \overset{b}{\hookrightarrow} pXX,\ qX \overset{1-b}{\hookrightarrow} p\varepsilon,$$

where both a, b are greater than $1/2$. Apparently, $[pXp] = [qXq] = 0$. Using results of [13] one can easily verify that $[pXq] = (1-a)/b$ and $[qXp] = (1-b)/a$. Thus

$[pX\uparrow] = (a+b-1)/b$ and $[qX\uparrow] = (a+b-1)/a$. Thus the stack symbols of $\Delta_\bullet$ are $\langle pXq\rangle, \langle qXp\rangle, \langle pX\uparrow\rangle, \langle qX\uparrow\rangle$. The transition rules of $\Delta_\bullet$ are:

$$\langle pXq\rangle \xrightarrow{1-b}_\bullet \langle qXp\rangle\langle pXq\rangle \quad \langle pXq\rangle \overset{b}{\hookrightarrow}_\bullet \varepsilon \quad \langle qXp\rangle \xrightarrow{1-a}_\bullet \langle pXq\rangle\langle qXp\rangle \quad \langle qXp\rangle \overset{a}{\hookrightarrow}_\bullet \varepsilon$$
$$\langle pX\uparrow\rangle \xrightarrow{1-b}_\bullet \langle qXp\rangle\langle pX\uparrow\rangle \quad \langle pX\uparrow\rangle \overset{b}{\hookrightarrow}_\bullet \langle qX\uparrow\rangle \quad \langle qX\uparrow\rangle \xrightarrow{1-a}_\bullet \langle pXq\rangle\langle qX\uparrow\rangle \quad \langle qX\uparrow\rangle \overset{a}{\hookrightarrow}_\bullet \langle pX\uparrow\rangle$$

As both a, b are greater than $1/2$, the resulting pBPA has a tendency to remove symbols rather than add symbols. Thus both $\langle pXq\rangle$ and $\langle qXp\rangle$ terminate with probability 1.

When studying long-run properties of pPDA (such as ω-regular properties or limit-average properties), one usually assumes that the runs are initiated in a configuration p_0X_0 which cannot terminate, i.e., $[p_0X_0\uparrow] = 1$. Under this assumption, the probability spaces over $Run[M_\Delta](p_0X_0)$ and $Run[M_{\Delta_\bullet}](\langle p_0X_0\uparrow\rangle)$ are "isomorphic" w.r.t. all properties that depend only on the control states and the top-of-the-stack symbols of the configurations visited along a run. This is formalized in our next proposition.

Proposition 5. *Let $p_0X_0 \in Q\times\Gamma$ such that $[p_0X_0\uparrow] = 1$. Then there is a partial function $\Upsilon : Run[M_\Delta](p_0X_0) \to Run[M_{\Delta_\bullet}](\langle p_0X_0\uparrow\rangle)$ such that for every $w \in Run[M_\Delta](p_0X_0)$, where $\Upsilon(w)$ is defined, and every $n \in \mathbb{N}$ we have the following: if $w(n) = qY\beta$, then $\Upsilon(w)(n) = \langle qY\dagger\rangle\gamma$, where $\dagger$ is either an element of Q or $\uparrow$. Further, for every measurable set of runs $R \subseteq Run[M_{\Delta_\bullet}](\langle p_0X_0\uparrow\rangle)$ we have that $\Upsilon^{-1}(R)$ is measurable and $\mathcal{P}(R) = \mathcal{P}(\Upsilon^{-1}(R))$.*

As for terminating runs, observe that the "terminating" symbols of the form $\langle pXq\rangle$ do not depend on the "nonterminating" symbols of the form $\langle pX\uparrow\rangle$, i.e., if we restrict $\Delta_\bullet$ just to terminating symbols, we again obtain a pBPA. A straightforward computation reveals the following proposition about terminating runs that is crucial for our results presented in the next section.

Proposition 6. *Let $pXq \in Q\times\Gamma\times Q$ and $[pXq] > 0$. Then almost all runs of $M_{\Delta_\bullet}$ initiated in $\langle pXq\rangle$ terminate, i.e., reach ε. Further, for all $n \in \mathbb{N}$ we have that*

$$\mathcal{P}(\mathbf{T}_{pX} = n \mid Run(pXq)) \quad = \quad \mathcal{P}(\mathbf{T}_{\langle pXq\rangle} = n \mid Run(\langle pXq\rangle))$$

Observe that this proposition, together with a very special form of rules in $\Delta_\bullet$, implies that all configurations reachable from a nonterminating configuration p_0X_0 have the form $\alpha\langle qY\uparrow\rangle$, where α terminates almost surely and $\langle qY\uparrow\rangle$ never terminates. It follows that such a pBPA can be transformed into a finite-state Markov chain (whose states are the nonterminating symbols) which is allowed to make recursive calls that almost surely terminate (using rules of the form $\langle pX\uparrow\rangle \hookrightarrow \langle rZq\rangle\langle qY\uparrow\rangle$). This observation is very useful when investigating the properties of nonterminating runs, and many of the existing results about pPDA can be substantially simplified using this result.

3.2 Analysis of pBPA

In this section we establish the promised tight tail bounds for termination probability. By virtue of Proposition 6, it suffices to analyze pBPA where each stack symbol terminates with probability 1. In what follows we assume that Δ is such a pBPA, and we also fix

an initial stack symbol X_0. For $X, Y \in \Gamma$, we say that X *depends directly on* Y, if there is a rule $X \hookrightarrow \alpha$ such that Y occurs in α. Further, we say that X *depends on* Y, if either X depends directly on Y, or X depends directly on a symbol $Z \in \Gamma$ which depends on Y. One can compute, in linear time, the directed acyclic graph (DAG) of strongly connected components (SCCs) of the dependence relation. The *height* of this DAG, denoted by h, is defined as the longest distance between a top SCC and a bottom SCC plus 1 (i.e., $h = 1$ if there is only one SCC). We can safely assume that all symbols on which X_0 does not depend were removed from Δ. We abbreviate $\mathcal{P}(\mathbf{T}_{X_0} \geq n \mid Run(X_0))$ to $\mathcal{P}(\mathbf{T}_{X_0} \geq n)$, and we use p_{min} to denote $\min\{p \mid X \xhookrightarrow{p} \alpha \text{ in } \Delta\}$. Here is our main result:

Theorem 7. *Let Δ be a pBPA with stack alphabet Γ where every stack symbol terminates with probability one. Assume that $X_0 \in \Gamma$ depends on all $X \in \Gamma \setminus \{X_0\}$, and let $p_{min} = \min\{p \mid X \xhookrightarrow{p} \alpha \text{ in } \Delta\}$. Then one of the following is true:*

(1) $\mathcal{P}(\mathbf{T}_{X_0} \geq 2^{|\Gamma|}) = 0$.

(2) $\mathbb{E}[\mathbf{T}_{X_0}]$ *is* **finite** *and for all $n \in \mathbb{N}$ with $n \geq 2\mathbb{E}[\mathbf{T}_{X_0}]$ we have that*

$$p_{min}^n \quad \leq \quad \mathcal{P}(\mathbf{T}_{X_0} \geq n) \quad \leq \quad \exp\left(1 - \frac{n}{8E_{max}^2}\right)$$

where $E_{max} = \max_{X \in \Gamma} \mathbb{E}[\mathbf{T}_X]$.

(3) $\mathbb{E}[\mathbf{T}_{X_0}]$ *is* **infinite** *and there is $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ we have that*

$$c/n \quad \leq \quad \mathcal{P}(\mathbf{T}_{X_0} \geq n) \quad \leq \quad d_1/n^{d_2}$$

where $d_1 = 18h|\Gamma|/p_{min}^{3|\Gamma|}$, and $d_2 = 1/(2^{h+1} - 2)$. Here, h is the height of the DAG of SCCs of the dependence relation, and c is a suitable positive constant depending on Δ.

More colloquially, Theorem 7 states that Δ satisfies either (1) or (2) or (3), where (1) is when Δ does not have any long terminating runs; and (2) resp. (3) is when the expected termination time is finite (resp. infinite) and the probability of performing a terminating run of length n decreases exponentially (resp. polynomially) in n.

One can effectively distinguish between the three cases set out in Theorem 7. More precisely, case (1) can be recognized in polynomial time by looking only at the structure of the pBPA, i.e., disregarding the probabilities. Determining whether $\mathbb{E}[\mathbf{T}_{X_0}]$ is finite or infinite can be done in polynomial space by employing the decision procedure for $\mathbb{E}[\mathbf{T}_{X_0}]$ and the results of [14,2]. This holds even if the transition probabilities of Δ are represented just symbolically by formulae of $ExTh(\mathbb{R})$ (see Proposition 3).

The proof of Theorem 7 is based on designing suitable martingales that are used to analyze the concentration of the termination probability. Recall that a *martingale* is an infinite sequence of random variables $m^{(0)}, m^{(1)}, \ldots$ such that, for all $i \in \mathbb{N}$, $\mathbb{E}[|m^{(i)}|] < \infty$, and $\mathbb{E}[m^{(i+1)} \mid m^{(1)}, \ldots, m^{(i)}] = m^{(i)}$ almost surely. If $|m^{(i)} - m^{(i-1)}| < c_i$ for all $i \in \mathbb{N}$, then we have the following *Azuma's inequality* (see, e.g., [27]):

$$\mathcal{P}(m^{(n)} - m^{(0)} \geq t) \quad \leq \quad \exp\left(\frac{-t^2}{2\sum_{k=1}^{n} c_k^2}\right)$$

Due to space restrictions we can only sketch the proof of Theorem 7 (see [8] for details).

Proof sketch for the upper bound of Theorem 7(2). Observe that if $\mathbb{E}[\mathbf{T}_{X_0}]$ is finite, then $\mathbb{E}[\mathbf{T}_Y]$ is finite for every $Y \in \Gamma$ (here we use the assumption that X_0 depends on Y). Further, for every configuration $\beta\gamma$ reachable from X_0 we have that $\mathbb{E}\left[\mathbf{T}_{\beta\gamma}\right] = \mathbb{E}\left[\mathbf{T}_\beta\right] + \mathbb{E}\left[\mathbf{T}_\gamma\right]$. Hence, $\mathbb{E}[\mathbf{T}_\alpha] < \infty$ for every $\alpha \in \Gamma^*$. Now observe that, for every $\alpha \in \Gamma^*$ such that $\alpha \neq \varepsilon$, performing one transition from α decreases the expected termination time by one on average (here we need that $\mathbb{E}[\mathbf{T}_\alpha] < \infty$ and α terminates with probability one). Let $w \in Run(X_0)$. We denote by $I(w)$ the maximal number $j \geq 0$ such that $w(j-1) \neq \varepsilon$. For every $i \geq 0$, we put

$$m^{(i)}(w) \quad = \quad \mathbb{E}[\mathbf{T}_{w(i)}] + \min\{i, I(w)\}$$

It is easy to see that $\mathbb{E}\left[m^{(i+1)} \mid m^{(i)}\right] = m^{(i)}$, i.e., $m^{(0)}, m^{(1)}, \ldots$ is a martingale. A full proof of this claim is given in [8].

Let $E_{max} = \max_{X \in \Gamma} \mathbb{E}[\mathbf{T}_X]$, and let $n \geq 2\mathbb{E}[\mathbf{T}_{X_0}]$. By applying Azuma's inequality we obtain

$$\mathcal{P}(m^{(n)} - \mathbb{E}[\mathbf{T}_{X_0}] \geq n - \mathbb{E}[\mathbf{T}_{X_0}]) \quad \leq \quad \exp\left(\frac{-(n - \mathbb{E}[\mathbf{T}_{X_0}])^2}{2\sum_{k=1}^{n}(2E_{max})^2}\right) \quad \leq \quad \exp\left(\frac{2\mathbb{E}[\mathbf{T}_{X_0}] - n}{8E_{max}^2}\right).$$

For every $w \in Run(X_0)$ we have that $w(n) \neq \varepsilon$ implies $m^{(n)} \geq n$. It follows:

$$\mathcal{P}(\mathbf{T}_{X_0} \geq n) \quad \leq \quad \mathcal{P}(m^{(n)} \geq n) \quad \leq \quad \exp\left(\frac{2\mathbb{E}[\mathbf{T}_{X_0}] - n}{8E_{max}^2}\right) \quad \leq \quad \exp\left(1 - \frac{n}{8E_{max}^2}\right).$$

Proof sketch for the upper bound of Theorem 7(3). Assume that $\mathbb{E}[\mathbf{T}_{X_0}]$ is infinite. To give some idea of the (quite involved) proof, let us first consider a simple pBPA Δ with $\Gamma = \{X\}$ and the rules $X \overset{1/2}{\hookrightarrow} XX$ and $X \overset{1/2}{\hookrightarrow} \varepsilon$. In fact, Δ is closely related to a simple random walk starting at 1, for which the time until it hits 0 can be exactly analyzed (see, e.g., [27]). Clearly, we have $h = |\Gamma| = 1$ and $p_{min} = 1/2$. Theorem 7(3) implies $\mathcal{P}(\mathbf{T}_X \geq n) \in O(1/\sqrt{n})$. Let us sketch why this upper bound holds.

Let $\theta > 0$, define $g(\theta) := \frac{1}{2} \cdot \exp(-\theta \cdot (-1)) + \frac{1}{2} \cdot \exp(-\theta \cdot (+1))$, and define for a run $w \in Run(X)$ the sequence

$$m_\theta^{(i)}(w) = \begin{cases} \exp(-\theta \cdot |w(i)|)/g(\theta)^i & \text{if } i = 0 \text{ or } w(i-1) \neq \varepsilon \\ m_\theta^{(i-1)}(w) & \text{otherwise.} \end{cases}$$

One can show (cf. [27]) that $m_\theta^{(0)}, m_\theta^{(1)}, \ldots$ is a martingale, i.e., $\mathbb{E}\left[m_\theta^{(i)} \mid m_\theta^{(i-1)}\right] = m_\theta^{(i-1)}$ for all $\theta > 0$. Our proof crucially depends on some analytic properties of the function $g : \mathbb{R} \to \mathbb{R}$: It is easy to verify that $1 = g(0) < g(\theta)$ for all $\theta > 0$, and $0 = g'(0)$, and $1 = g''(0)$. One can show that Doob's Optional-Stopping Theorem (see Theorem 10.10 (ii) of [27]) applies, which implies $m_\theta^{(0)} = \mathbb{E}\left[m_\theta^{(\mathbf{T}_X)}\right]$. It follows that for all $n \in \mathbb{N}$ and $\theta > 0$ we have that

$$\exp(-\theta) = m_\theta^{(0)} = \mathbb{E}\left[m_\theta^{(\mathbf{T}_X)}\right] = \mathbb{E}\left[g(\theta)^{-\mathbf{T}_X}\right] = \sum_{i=0}^{\infty} \mathcal{P}(\mathbf{T}_X = i) \cdot g(\theta)^{-i}$$

$$\leq \sum_{i=0}^{n-1} \mathcal{P}(\mathbf{T}_X = i) \cdot 1 + \sum_{i=n}^{\infty} \mathcal{P}(\mathbf{T}_X = i) \cdot g(\theta)^{-n} = 1 - \mathcal{P}(\mathbf{T}_X \geq n) + \mathcal{P}(\mathbf{T}_X \geq n) \cdot g(\theta)^{-n}$$

Rearranging this inequality yields $\mathcal{P}(\mathbf{T}_X \geq n) \leq \frac{1-\exp(-\theta)}{1-g(\theta)^{-n}}$, from which one obtains, setting $\theta := 1/\sqrt{n}$, and using the mentioned properties of g and several applications of l'Hopital's rule, that $\mathcal{P}(\mathbf{T}_X \geq n) \in \mathcal{O}(1/\sqrt{n})$.

Next we sketch how we generalize this proof to pBPA that consist of only one SCC, but have more than one stack symbol. In this case, the term $|w(i)|$ in the definition of $m_\theta^{(i)}(w)$ needs to be replaced by the sum of *weights* of the symbols in $w(i)$. Each $Y \in \Gamma$ has a weight which is drawn from the dominant eigenvector of a certain matrix, which is characteristic for Δ. Perron-Frobenius theory guarantees the existence of a suitable weight vector $\boldsymbol{u} \in \mathbb{R}_+^\Gamma$. The function g consequently needs to be replaced by a function g_Y for each $Y \in \Gamma$. We need to keep the property that $g_Y''(0) > 0$. Intuitively, this means that Δ must have, for each $Y \in \Gamma$, a rule $Y \hookrightarrow \alpha$ such that Y and α have different weights. This can be accomplished by transforming Δ into a certain normal form.

Finally, we sketch how the proof is generalized to pBPA with more than one SCC. For simplicity, assume that Δ has only two stack symbols, say X and Y, where X depends on Y, but Y does not depend on X. Let us change the execution order of pBPA as follows: whenever a rule with $\alpha \in \Gamma^*$ on the right hand side fires, then all X-symbols in α are added on top of the stack, but all Y-symbols are added at the *bottom* of the stack. This change does not influence the termination time of pBPA, but it allows to decompose runs into two phases: an X-phase where X-rules are executed which may produce Y-symbols or further X-symbols; and a Y-phase where Y-rules are executed which may produce further Y-symbols but no X-symbols, because Y does not depend on X. Arguing only qualitatively, assume that $\mathbf{T}_X$ is "large". Then either (a) the X-phase is "long" or (b) the X-phase is "short", but the Y-phase is "long". For the probability of event (a) one can give an upper bound using the bound for one SCC, because the produced Y-symbols can be ignored. For event (b), observe that if the X-phase is short, then only few Y-symbols can be created during the X-phase. For a bound on the probability of event (b) we need a bound on the probability that a pBPA with one SCC and a "short" initial configuration takes a "long" time to terminate. The previously sketched proof for an initial configuration with a single stack symbol can be suitably generalized to handle other "short" configurations. The details can be found in [8].

Finally, the following proposition shows that the upper bound in Theorem 7 (3) cannot be substantially tightened.

Proposition 8. *Let Δ_h be the pBPA with $\Gamma_h = \{X_1, \ldots, X_h\}$ and the following rules:*

$$X_h \xhookrightarrow{1/2} X_h X_h\,,\ X_h \xhookrightarrow{1/2} X_{h-1}\,,\ \ldots\,,\ X_2 \xhookrightarrow{1/2} X_2 X_2\,,\ X_2 \xhookrightarrow{1/2} X_1\,,\ X_1 \xhookrightarrow{1/2} X_1 X_1\,,\ X_1 \xhookrightarrow{1/2} \varepsilon$$

Then $[X_h] = 1$, $\mathbb{E}[\mathbf{T}_{X_h}] = \infty$, and there is $c_h > 0$ with $\mathcal{P}(\mathbf{T}_{X_h} \geq n) \geq c_h \cdot n^{-1/2^h}$ for all $n \geq 1$.

4 Conclusions and Future Work

We have provided a reduction from stateful to stateless pPDA which gives new insights into the theory of pPDA and at the same time simplifies it substantially. We have used this reduction and martingale theory to exhibit a dichotomy result that precisely characterizes the distribution of the termination time in terms of its expected value.

Although the bounds presented in this paper are asymptotically optimal, there is still space for improvements. We conjecture that the lower bound of Theorem 7 (3) can be

strengthened to $\Omega(1/\sqrt{n})$. We also conjecture that our results can be extended to more general reward-based models, where each configuration is assigned a nonnegative reward and the total reward accumulated in a given service is considered instead of its length. This is particularly challenging if the rewards are unbounded (for example, the reward assigned to a given configuration may correspond to the total memory allocated by the procedures in the current call stack). Full answers to these questions would generalize some of the existing deep results about simpler models, and probably reveal an even richer underlying theory of pPDA which is still undiscovered.

Acknowledgment. The authors thank Javier Esparza for useful suggestions.

References

1. Bini, D., Latouche, G., Meini, B.: Numerical methods for Structured Markov Chains. Oxford University Press, Oxford (2005)
2. Brázdil, T.: Verification of Probabilistic Recursive Sequential Programs. PhD thesis, Masaryk University, Faculty of Informatics (2007)
3. Brázdil, T., Brožek, V., Holeček, J., Kučera, A.: Discounted properties of probabilistic pushdown automata. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 230–242. Springer, Heidelberg (2008)
4. Brázdil, T., Brožek, V., Etessami, K.: One-counter stochastic games. In: Proceedings of ST&TCS 2010. Leibniz International Proceedings in Informatics, vol. 8, pp. 108–119. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2010)
5. Brázdil, T., Brožek, V., Etessami, K., Kučera, A., Wojtczak, D.: One-counter Markov decision processes. In: Proceedings of SODA 2010, pp. 863–874. SIAM, Philadelphia (2010)
6. Brázdil, T., Esparza, J., Kučera, A.: Analysis and prediction of the long-run behavior of probabilistic sequential programs with recursion. In: Proceedings of FOCS 2005, pp. 521–530. IEEE Computer Society Press, Los Alamitos (2005)
7. Brázdil, T., Kiefer, S., Kučera, A.: Efficient analysis of probabilistic programs with an unbounded counter. CoRR, abs/1102.2529 (2011)
8. Brázdil, T., Kiefer, S., Kučera, A., Hutařová Vařeková, I.: Runtime analysis of probabilistic programs with unbounded recursion. CoRR, abs/1007.1710 (2010)
9. Canny, J.: Some algebraic and geometric computations in PSPACE. In: Proceedings of STOC 1988, pp. 460–467. ACM Press, New York (1988)
10. Dubhashi, D.P., Panconesi, A.: Concentration of Measure for the Analysis of Randomized Algorithms. Cambridge University Press, Cambridge (2009)
11. Esparza, J., Kiefer, S., Luttenberger, M.: Convergence thresholds of Newton's method for monotone polynomial equations. In: STACS 2008, pp. 289–300 (2008)
12. Esparza, J., Kiefer, S., Luttenberger, M.: Computing the least fixed point of positive polynomial systems. SIAM Journal on Computing 39(6), 2282–2335 (2010)
13. Esparza, J., Kučera, A., Mayr, R.: Model-checking probabilistic pushdown automata. In: Proceedings of LICS 2004, pp. 12–21. IEEE Computer Society Press, Los Alamitos (2004)
14. Esparza, J., Kučera, A., Mayr, R.: Quantitative analysis of probabilistic pushdown automata: Expectations and variances. In: Proceedings of LICS 2005, pp. 117–126. IEEE Computer Society Press, Los Alamitos (2005)
15. Etessami, K., Wojtczak, D., Yannakakis, M.: Quasi-birth-death processes, tree-like QBDs, probabilistic 1-counter automata, and pushdown systems. In: Proceedings of 5th Int. Conf. on Quantitative Evaluation of Systems (QEST 2008). IEEE Computer Society Press, Los Alamitos (2008)

16. Etessami, K., Yannakakis, M.: Algorithmic verification of recursive probabilistic systems. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 253–270. Springer, Heidelberg (2005)
17. Etessami, K., Yannakakis, M.: Checking LTL properties of recursive Markov chains. In: Proceedings of 2nd Int. Conf. on Quantitative Evaluation of Systems (QEST 2005), pp. 155–165. IEEE Computer Society Press, Los Alamitos (2005)
18. Etessami, K., Yannakakis, M.: Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. Journal of the Association for Computing Machinery 56 (2009)
19. Harris, T.E.: The Theory of Branching Processes. Springer, Heidelberg (1963)
20. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
21. Kiefer, S., Luttenberger, M., Esparza, J.: On the convergence of Newton's method for monotone systems of polynomial equations. In: STOC 2007, pp. 217–226 (2007)
22. Latouche, G., Ramaswami, V.: Introduction to Matrix Analytic Methods in Stochastic Modeling. ASA-SIAM series on statistics and applied probability (1999)
23. Manning, C., Schütze, H.: Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge (1999)
24. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (2006)
25. Pakes, A.G.: Some limit theorems for the total progeny of a branching process. Advances in Applied Probability 3(1), 176–192 (1971)
26. Quine, M.P., Szczotka, W.: Generalisations of the Bienayme-Galton-Watson branching process via its representation as an embedded random walk. The Annals of Applied Probability 4(4), 1206–1222 (1994)
27. Williams, D.: Probability with Martingales. Cambridge University Press, Cambridge (1991)

Approximating the Termination Value of One-Counter MDPs and Stochastic Games

Tomáš Brázdil[1,⋆], Václav Brožek[2,⋆⋆], Kousha Etessami[2], and Antonín Kučera[1,⋆]

[1] Faculty of Informatics, Masaryk University
{xbrazdil,tony}@fi.muni.cz
[2] School of Informatics, University of Edinburgh
{vbrozek,kousha}@inf.ed.ac.uk

Abstract. One-counter MDPs (OC-MDPs) and one-counter simple stochastic games (OC-SSGs) are 1-player, and 2-player turn-based zero-sum, stochastic games played on the transition graph of classic one-counter automata (equivalently, pushdown automata with a 1-letter stack alphabet). A key objective for the analysis and verification of these games is the *termination* objective, where the players aim to maximize (minimize, respectively) the probability of hitting counter value 0, starting at a given control state and given counter value.

Recently [4,2], we studied *qualitative* decision problems ("is the optimal termination value = 1?") for OC-MDPs (and OC-SSGs) and showed them to be decidable in P-time (in NP∩coNP, respectively). However, *quantitative* decision and approximation problems ("is the optimal termination value $\geq p$", or "approximate the termination value within ε") are far more challenging. This is so in part because optimal strategies may not exist, and because even when they do exist they can have a highly non-trivial structure. It thus remained open even whether any of these quantitative termination problems are computable.

In this paper we show that all quantitative *approximation* problems for the termination value for OC-MDPs and OC-SSGs are computable. Specifically, given a OC-SSG, and given $\varepsilon > 0$, we can compute a value v that approximates the value of the OC-SSG termination game within additive error ε, and furthermore we can compute ε-optimal strategies for both players in the game.

A key ingredient in our proofs is a subtle martingale, derived from solving certain LPs that we can associate with a maximizing OC-MDP. An application of Azuma's inequality on these martingales yields a computable bound for the "wealth" at which a "rich person's strategy" becomes ε-optimal for OC-MDPs.

1 Introduction

In recent years, there has been substantial research done to understand the computational complexity of analysis and verification problems for classes of finitely-presented but infinite-state stochastic models, MDPs, and stochastic games, whose transition

⋆ Tomáš Brázdil and Antonín Kučera are supported by the Institute for Theoretical Computer Science (ITI), project No. 1M0545, and by the Czech Science Foundation, grant No. P202/10/1469.

⋆⋆ Václav Brožek is supported by the Newton International Fellowship of Royal Society.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 332–343, 2011.

graphs arise from basic infinite-state automata-theoretic models, including: context-free processes, one-counter processes, and pushdown processes. It turns out these models are intimately related to important stochastic processes studied extensively in applied probability theory. In particular, one-counter probabilistic automata are basically equivalent to (discrete-time) quasi-birth-death processes (QBDs) (see [8]), which are heavily studied in queuing theory and performance evaluation as a basic model of an unbounded queue with multiple states (phases). It is very natural to extend these purely probabilistic models to MDPs and games, to model adversarial queuing scenarios.

In this paper we continue this work by studying quantitative *approximation* problems for **one-counter MDPs (OC-MDPs)** and **one-counter simple stochastic games (OC-SSGs)**, which are 1-player, and turn-based zero-sum 2-player, 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. 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) OC-MDP. Fixing strategies for the two players yields a countable state Markov chain and thus a probability space of infinite runs (trajectories).

A central objective for the analysis and verification of OC-SSGs, is the *termination* objective: 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). From well know fact, it follows that these games are *determined*, meaning they have a *value*, v, such that for every $\varepsilon > 0$, player Max (Min) has a strategy that ensures the objective is satisfied with probability at least $v - \varepsilon$ (at most $v + \varepsilon$, respectively), regardless of what the other player does. This value can be *irrational* even when the input data contains only rational probabilities, and this is so even in the purely stochastic case of QBDs without players ([8]).

A special subclass of OC-MDPs, called *solvency games*, was studied in [1] as a simple model of risk-averse investment. Solvency games correspond to OC-MDPs where there is only one control state, but there are multiple actions that change the counter value ("wealth"), possibly by more than 1 per transition, according to a finite support probability distribution on the integers associated with each action. The goal is to minimize the probability of going bankrupt, starting with a given positive wealth. It is not hard to see that these are subsumed by minimizing OC-MDPs (see [4]). It was shown in [1] that if the solvency game satisfies a number of restrictive assumptions (in particular, on the eigenvalues of a matrix associated with the game), then an optimal "rich person's" strategy (which does the same action whenever the wealth is large enough) can be computed for it (in exponential time). They showed such strategies are not optimal for unrestricted solvency games and left the unrestricted case unresolved in [1].

We can classify analysis problems for OC-MDPs and OC-SSGs into two kinds. *Quantitative* analyses, which include: "is the game value at least/at most p" for a given $p \in [0, 1]$; or "approximate the game value" to within a desired additive error $\varepsilon > 0$. We

can also restrict ourselves to *qualitative* analyses, which asks "is the game value = 1? = 0?".[1] We are also interested in strategies (e.g., memoryless, etc.) that achieve these.

In recent work [4,2], we have studied *qualitative* termination problems for OC-SSGs. For both *maximizing* and *minimizing* OC-MDPs, we showed that these problems are decidable in P-time, using linear programming, connections to the theory of random walks on integers, and other MDP objectives. For OC-SSGs, we showed the qualitative termination problem "is the termination value = 1?" is in NP ∩ coNP. This problem is already as hard as Condon's quantitative termination problem for finite-state SSGs.

However we left open, as the main open question, the computability of *quantitative* termination problems for OC-MDPs and OC-SSGs. In this paper, we resolve positively the computability of all quantitative *approximation* problems associated with OC-MDPs and OC-SSGs. Note that, in some sense, approximation of the termination value in the setting of OC-MDPs and OC-SSGs can not be avoided. This is so not only because the value can be irrational, but because (see [3]) for maximizing OC-MDPs there need not exist any optimal strategy for maximizing the termination probability, only ε-optimal ones (whereas Min does have an optimal strategy in OC-SSGs). Moreover, even for minimizing OC-MDPs, where optimal strategies do exist, they can have a very complicated structure. In particular, as already mentioned for solvency games, there need not exist any "rich person's" strategy that can ignore the counter value when it is larger than some finite $N \geq 0$.

Nevertheless, we show all these difficulties can be overcome when the goal is to *approximate* the termination value of OC-SSGs and to compute ε-optimal strategies. Our main theorem is the following:

Theorem 1 (ε-approximation of OC-SSG termination value). *Given as input: a OC-SSG, $\mathcal{G}$, an initial control state s, an initial counter value $j > 0$, and a (rational) approximation threshold $\varepsilon > 0$, there is an algorithm that computes a rational number, v', such that $|v' - v^*| < \varepsilon$, where v^* is the value of the OC-SSG termination game on G, starting in configuration (s, j). Moreover, there is an algorithm that computes ε-optimal strategies for both players in the OC-SSG termination game. These algorithms run in exponential time in the encoding size of a 1-player OC-SSG, i.e., a OC-MDP, and in polynomial time in $\log(1/\varepsilon)$ and $\log(j)$. In the case of 2-player OC-SSGs, the algorithms run in nondeterministic exponential time in the encoding size of the OC-SSG.*

We now outline our basic strategy for proving this theorem. Consider the case of maximizing OC-MDPs, and suppose we would like to approximate the optimal termination probability, starting at state q and counter value i. Intuitively, it is not hard to see that as the counter value goes to infinity, except for some basic cases that we can detect and eliminate in polynomial time, the optimal probability of termination starting at a state q begins to approach the optimal probability of forcing the counter to have a lim inf value $= -\infty$. But we can compute this optimal value, and an optimal strategy for it, based on results in our prior work [4,2]. Of particular importance are the set of states T from which this value is 1. For a given $\varepsilon > 0$, we need to compute a bound N on the counter value, such that for any state q, and all counter values $N' > N$, the optimal termination

[1] The problem "is the termination value = 0?" is easier, and can be solved in polynomial time without even looking at the probabilities labeling the transitions of the OC-SSG.

probability starting at (q, N') is at most ε away from the optimal probability for the counter to have lim inf value $= -\infty$. *A priori* it is not at all clear whether such a bound N is computable, although it is clear that N exists. To show that it is computable, we employ a subtle (sub)martingale, derived from solving a certain linear programming problem associated with a given OC-MDP. By applying Azuma's inequality on this martingale, we are able to show there are computable values $c < 1$, and $h \geq 0$, such that for all $i > h$, starting from a state q and counter value i, the optimal probability of both terminating and not encountering any state from which with probability 1 the player can force the lim inf counter value to go to $-\infty$, is at most $c^i/(1-c)$. Thus, the optimal termination probability approaches from above the optimal probability of forcing the lim inf counter value to be $-\infty$, and the difference between these two values is exponentially small in i, with a computable base c. This martingale argument extends to OC-MDPs an argument recently used in [7] for analyzing purely probabilistic one-counter automata (i.e., QBDs).

These bounds allow us to reduce the problem of approximating the termination value to the reachability problem for an exponentially larger finite-state MDP, which we can solve (in exponential time) using linear programming. The case for general OC-SSGs and minimizing OC-MDPs turns out to follow a similar line of argument, reducing the essential problem to the case of maximizing OC-MDPs. In terms of complexity, the OC-SSG case requires "guessing" an appropriate (albeit, exponential-sized) strategy, whereas the relevant exponential-sized strategy can be computed in deterministic exponential time for OC-MDPs. So our approximation algorithms run in exponential time for OC-MDPs and nondeterministic exponential time for OC-SSGs.

Open problems. An obvious remaining open problem is to obtain better complexity bounds for OC-MDPs. We know of no non-trivial lower bounds for OC-MDP approximation problems. Our results also leave open the decidability of the quantitative termination *decision* problem for OC-MDPs and OC-SSGs, which asks: "is the termination value $\geq p$?" for a given rational probability p. Furthermore, our results leave open computability for approximating the value of *selective termination* objectives for OC-MDPs, where the goal is to terminate (reach counter value 0) in a specific subset of the control states. Qualitative versions of selective termination problems were studied in [4,2].

Related work. As noted, 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 studied in [9], where it was shown that many interesting computational problems, including any nontrivial approximation of the termination value for general RSSGs and RMDPs is undecidable, as are qualitative termination problems. It was also shown 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 NP$\cap$coNP ([10]), while quantitative termination problems are decidable in PSPACE. Solving termination objectives is a key ingredient for many more general

analyses and model checking problems for such stochastic games (see, e.g., [5,6]). OC-SSGs are incompatible with stochastic context-free games. Specifically, for OC-SSGs, the number of stack symbols is bounded by 1, instead of the number of control states.

MDP variants of QBDs, essentially equivalent to OC-MDPs, have been considered in the queueing theory and stochastic modeling literature, see [14,12]. However, in order to keep their analyses tractable, these works perform a naive finite-state "approximation" by cutting off the value of the counter at an arbitrary finite value N, and adding *dead-end absorbing* states for counter values higher than N. Doing this can radically alter the behavior of the model, even for purely probabilistic QBDs, and these authors establish no rigorous approximation bounds for their models. In a sense, our work can be seen as a much more careful and rigorous approach to finite approximation, employing at the boundary other objectives like maximizing the probability that the lim inf counter value $= -\infty$. Unlike the prior work we establish rigorous bounds on how well our finite-state model approximates the original infinite OC-MDP.

2 Definitions

We assume familiarity with basic notions from probability theory. We call a probability distribution f over a discrete set, A, *positive* if $f(a) > 0$ for all $a \in A$, and *Dirac* if $f(a) = 1$ for some $a \in A$.

Definition 1 (SSG). *A* simple stochastic game (SSG) *is a tuple* $\mathcal{G} = (S, (S_0, S_1, S_2), \rightsquigarrow, Prob)$*, consisting of a countable set of* states*,* S*, partitioned into the set* S_0 *of* stochastic *states, and sets* S_1*,* S_2 *of states owned by Player 1 (Max) and 2 (Min), respectively. The* edge relation $\rightsquigarrow \subseteq S \times S$ *is total, i.e., for every* $r \in S$ *there is* $s \in S$ *such that* $r \rightsquigarrow s$*. Finally, Prob assigns to every* $s \in S_0$ *a positive probability distribution over outgoing edges. If* $S_2 = \emptyset$*, we call the SSG a* maximizing *Markov Decision Processes (MDP). If* $S_1 = \emptyset$ *we call it a* minimizing *MDP.*

A *finite path* is a sequence $w = s_0 s_1 \cdots s_n$ of states such that $s_i \rightsquigarrow s_{i+1}$ for all $i, 0 \leq i < n$. We write $len(w) = n$ for the length of the path. A *run*, ω, is an infinite sequence of states every finite prefix of which is a path. For a finite path, w, we denote by $Run(w)$ the set of runs having w as a prefix. These generate the standard σ-algebra on the set of runs.

Definition 2 (OC-SSG). *A* one-counter SSG (OC-SSG)*,* $\mathcal{A} = (Q, (Q_0, Q_1, Q_2), \delta, P)$*, consists of a finite non-empty set of* control states*,* Q*, partitioned into stochastic and players' states, as in the case of SSGs, a set of* transition rules $\delta \subseteq Q \times \{+1, 0, -1\} \times Q$ *such that* $\delta(q) := \{(q, i, r) \in \delta\} \neq \emptyset$ *for all* $q \in Q$*, and* $P = \{P_q\}_{q \in Q_0}$ *where* P_q *is a positive rational probability distribution over* $\delta(q)$ *for all* $q \in Q_0$*.*

Purely for convenience, we assume that for each pair $q, r \in Q$ there is at most one i such that $(q, i, r) \in \delta$ (this is clearly w.l.o.g., by adding suitable auxiliary states to Q). By $||\mathcal{A}|| := |Q| + |\delta| + ||P||$ we denote the encoding size of $\mathcal{A}$, where $||P||$ is the sum of the number of bits needed to encode the numerator and denominator of $P_q(\varrho)$ for all $q \in Q$ and $\varrho \in \delta$. The set of all *configurations* is $C := \{(q, i) \mid q \in Q, i \geq 0\}$. Again, maximizing and minimizing OC-MDPs are defined as analogous subclasses of OC-SSGs.

To $\mathcal{A}$ we associate an infinite-state SSG $\mathcal{A}^\infty = (C, (C_0, C_1, C_2), \rightarrow, Prob)$, where the partition of C is defined by $(q,i) \in C_0$ iff $q \in Q_0$, and similarly for the players. The edges are defined by $(q,i) \rightarrow (r,j)$ iff either $i > 0$ and $(q, j-i, r) \in \delta$, or $i = j = 0$ and $q = r$. The probability assignment $Prob$ is derived naturally from P.

By forgetting the counter values, the OC-SSG $\mathcal{A}$ also describes a finite-state SSG $\mathcal{G}_\mathcal{A} = (Q, (Q_0, Q_1, Q_2), \rightsquigarrow, Prob')$. Here $q \rightsquigarrow r$ iff $(q,i,r) \in \delta$ for some i, and $Prob'$ is derived in the obvious way from P by forgetting the counter changes. If $\mathcal{A}$ is a OC-MDP, both $\mathcal{G}_\mathcal{A}$ and $\mathcal{A}^\infty$ are MDPs.

Strategies and Probability. Let $\mathcal{G}$ be a SSG. A *history* is a finite path in $\mathcal{G}$. A *strategy* for Player 1 in $\mathcal{G}$, is a function assigning to each history ending in a state from S_1 a distribution on edges leaving the last state of the history. A strategy is *pure* if it always assigns a Dirac distribution, i.e., one which assigns 1 to one edge and 0 to the others. A strategy, σ, is *memoryless* if $\sigma(w) = \sigma(s)$ where s is the last state of a history w. Assume that $\mathcal{G} = \mathcal{A}^\infty$ for some OC-SSG $\mathcal{A}$. Then a strategy, σ, is *counterless* if it is memoryless and $\sigma((q,i)) = \sigma((q,1))$ for all $i \geq 1$. Observe that every strategy, σ, for $\mathcal{G}_\mathcal{A}$ gives a unique strategy, σ', for $\mathcal{A}^\infty$; the strategy σ' just forgets the counter values in the history and plays as σ. This correspondence is bijective when restricted to memoryless strategies in $\mathcal{G}_\mathcal{A}$ and counterless strategies in $\mathcal{A}^\infty$. We will use this correspondence implicitly throughout the paper. Strategies for Player 2 are defined analogously.

Fixing a pair (σ, π) of strategies for Player 1 and 2, respectively, and an initial state, s, we obtain in a standard way a probability measure $\mathbb{P}_s^{\sigma,\pi}(\cdot)$ on the subspace of runs starting in s. For SSGs of the form $\mathcal{A}^\infty$ for some OC-SSG, $\mathcal{A}$, we consider two sequences of random variables, $\{C^{(i)}\}_{i\geq 0}$ and $\{S^{(i)}\}_{i\geq 0}$, returning the height of the counter, and the control state after completing i transitions.

For a SSG, $\mathcal{G}$, an *objective*, R, is for us a Borel subset of runs in $\mathcal{G}$. Player 1 is trying to *maximize* the probability of R, while player 2 is trying to *minimize* it. We say that $(\mathcal{G}, R)$ is *determined* if for every state s of $\mathcal{G}$ we have that $\sup_\sigma \inf_\pi \mathbb{P}_s^{\sigma,\pi}(R) = \inf_\pi \sup_\sigma \mathbb{P}_s^{\sigma,\pi}(R)$. If $(\mathcal{G}, R)$ is determined, then for every state s of $\mathcal{G}$, the above equality defines the *value* of s, denoted by $\mathrm{Val}(R, s)$. For a given $\varepsilon \geq 0$, a strategy, σ^*, of Player 1 is *ε-optimal* in s, if $\mathbb{P}_s^{\sigma^*,\pi}(R) \geq \mathrm{Val}(R,s) - \varepsilon$ for every strategy π of Player 2. An ε-optimal strategy for Player 2 is defined analogously. 0-optimal strategies are called *optimal*. Note that $(\mathcal{G}, R)$ is determined iff both players have ε-optimal strategies for every $\varepsilon > 0$.

Termination Objective. Let $\mathcal{A}$ be a OC-SSG. A run in $\mathcal{A}^\infty$ *terminates* if it contains a configuration of the form $(q, 0)$. The *termination objective* is the set of all terminating runs, and is denoted Term. OC-SSG termination games are determined (see [2]).

3 Main Result

Theorem 1 (Main). *Given an OC-SSG, $\mathcal{A}$, a configuration, (q,i), and a rational $\varepsilon > 0$, there is an algorithm that computes a rational number, ν, such that $|\mathrm{Val}(\mathrm{Term}, (q,i)) - \nu| \leq \varepsilon$, and strategies σ, π for both players that are ε-optimal starting in (q,i). The algorithm runs in nondeterministic time exponential in $||\mathcal{A}||$ and polynomial in $\log(i)$ and $\log(1/\varepsilon)$. If $\mathcal{A}$ is an OC-MDP, then the algorithm runs in deterministic time exponential in $||\mathcal{A}||$ and polynomial in $\log(1/\varepsilon)$ and $\log(i)$.*

3.1 Proof Sketch

We now sketch the main ideas in the proof of Theorem 1. First, observe that for all $q \in Q$ and $i \leq j$ we have that $\mathrm{Val}(\mathrm{Term}, (q, i)) \geq \mathrm{Val}(\mathrm{Term}, (q, j)) \geq 0$. Let

$$\mu_q \coloneqq \lim_{i \to \infty} \mathrm{Val}(\mathrm{Term}, (q, i)).$$

Since $\mu_q \leq \mathrm{Val}(\mathrm{Term}, (q, i))$ for an arbitrarily large i, Player 1 should be able to decrease the counter by an arbitrary value with probability at least μ_q, no matter what Player 2 does. The objective of "decreasing the counter by an arbitrary value" cannot be formalized directly on $\mathcal{A}^\infty$, because the counter cannot become negative in the configurations of $\mathcal{A}^\infty$. Instead, we formalize this objective on $\mathcal{G}_\mathcal{A}$, extended with rewards on transitions. These rewards are precisely the counter changes, which were left out from the transition graph, i.e., each transition $q \rightsquigarrow r$ generated by a rule (q, i, r) of $\mathcal{A}$ has reward i. This allows us to define a sequence, $\{R^{(i)}\}_{i \geq 0}$, of random variables for runs in $\mathcal{G}_\mathcal{A}$, where $R^{(i)}$ returns the sum total of rewards accumulated during the first i steps. Note that $R^{(i)}$ may be negative, unlike the r.v. $C^{(i)}$. The considered objective then corresponds to the event $\mathit{LimInf}(= -\infty)$ consisting of all runs w in $\mathcal{G}_\mathcal{A}$ such that $\liminf_{i \to \infty} R^{(i)}(w) = -\infty$. These games are determined. For every $q \in Q$, let

$$\nu_q \coloneqq \mathrm{Val}(\mathit{LimInf}(= -\infty), q).$$

One intuitively expects that $\mu_q = \nu_q$, and we show that this is indeed the case. Further, it was shown in [4,2] that ν_q is rational and computable in non-deterministic time polynomial in $||\mathcal{A}||$. Moreover, both players have optimal pure memoryless strategies (σ^*, π^*) in $\mathcal{G}_\mathcal{A}$, computable in non-deterministic polynomial time. For MDPs, both the value ν_q and the optimal strategies can be computed in deterministic time polynomial in $||\mathcal{A}||$.

Since $\mu_q = \nu_q$, there is a sufficiently large N such that $\mathrm{Val}(\mathrm{Term}, (q, i)) - \nu_q \leq \varepsilon$ for all $q \in Q$ and $i \geq N$. We show that an upper bound on N is computable, which is at most exponential in $||\mathcal{A}||$ and polynomial in $\log(1/\varepsilon)$, in Section 3.2. As we shall see, this part is highly non-trivial. For all configurations (q, i), where $i \geq N$, the value $\mathrm{Val}(\mathrm{Term}, (q, i))$ can be approximated by ν_q, and both players can use the optimal strategies (σ^*, π^*) for the $\mathit{LimInf}(= -\infty)$ objective (which are "translated" into the corresponding counterless strategies in $\mathcal{A}^\infty$; cf. Section 2). For the remaining configurations (q, i), where $i < N$, we consider a finite-state SSG obtained by restricting $\mathcal{A}^\infty$ to configurations with counter between 0 and N, extended by two fresh stochastic states s_0, s_1 with self-loops. All configurations of the form $(q, 0)$ have only one outgoing edge leading to s_0, and all configurations of the form (q, N) can enter either s_0 with probability ν_q, or s_1 with probability $1 - \nu_q$. In this finite-state game, we compute the values and optimal strategies for the reachability objective, where the set of target states is $\{s_0\}$. This can be done in non-deterministic time polynomial in the size of the game (i.e., exponential in $||\mathcal{A}||$). If $\mathcal{A}$ is an OC-MDP, then the values and optimal strategies can be computed in deterministic polynomial time in the size of the MDP (i.e., exponential in $||\mathcal{A}||$) by linear programming (this applies both to the "maximizing" and the "minimizing" OC-MDP). Thus, we obtain the required approximations of $\mathrm{Val}(\mathrm{Term}, (q, i))$ for $i < N$, and the associated ε-optimal strategies.

Technically, we first consider the simpler case when $\mathcal{A}$ is a "maximizing" OC-MDP (Section 3.2). The general case is then obtained simply by computing the optimal counterless strategy π^* for the $LimInf(=-\infty)$ objective in $\mathcal{G}_{\mathcal{A}}$, and "applying" this strategy to resolve the choices of Player 2 in $\mathcal{A}^\infty$ (again, note that π^* corresponds to a counterless strategy in $\mathcal{A}^\infty$). Thus, we obtain an OC-MDP $\mathcal{A}'$ and apply the result of Section 3.2.

3.2 Bounding Counter Value N for Maximizing OC-MDPs

In this section we consider a maximizing OC-MDP $\mathcal{A} = (Q, (Q_0, Q_1), \delta, P)$. The *maximum termination value* is $\mathrm{Val}(\mathrm{Term}, (q,i)) = \sup_\sigma \mathbb{P}^\sigma_{(q,i)}(\mathrm{Term})$.

For a $q \in Q$ we set $\nu_q := \sup_\sigma \mathbb{P}^\sigma_q(LimInf(=-\infty))$. Given $\mathcal{A}$, and $\varepsilon > 0$, we show here how to obtain a computable (exponential) bound on a number N such that $\left|\mathrm{Val}(\mathrm{Term}, (q,i)) - \nu_q\right| < \varepsilon$ for all $i \geq N$. Thus, by the arguments described in the Section 3, once we have such a computable bound on N, we have an algorithm for approximating $\mathrm{Val}(\mathrm{Term}, (q,i))$. We denote by T the set of all states q with $\nu_q = 1$.

Fact 2 (cf. [4]). *The number ν_q is the max. probability of reaching T from q in $\mathcal{G}_{\mathcal{A}}$:*

$$\nu_q = \sup_\sigma \mathbb{P}^\sigma_q(reach\ T) = \max_\sigma \mathbb{P}^\sigma_q(reach\ T).$$

Claim. $\forall q \in Q : \forall i \geq 0 : \nu_q \leq \mathrm{Val}(\mathrm{Term}, (q,i)) \leq \sup_\sigma \mathbb{P}^\sigma_{(q,i)}(\mathrm{Term} \cap \text{not reach } T) + \nu_q$.

Proof. The first inequality is easy. By [4, Theorem 12], $\mathrm{Val}(\mathrm{Term}, (q,i)) = 1$ for all $q \in T$, $i \geq 0$, proving the second inequality. □

Lemma 1. *Given a maximizing OC-MDP, $\mathcal{A}$, one can compute a rational constant $c < 1$, and an integer $h \geq 0$ such that for all $i \geq h$ and $q \in Q$:* $\sup_\sigma \mathbb{P}^\sigma_{(q,i)}(\mathrm{Term} \cap \textit{not reach } T) \leq \frac{c^i}{1-c}$.
Moreover, $c \in \exp(1/2^{||\mathcal{A}||^{O(1)}})$ and $h \in \exp(||\mathcal{A}||^{O(1)})$.

Observe that this allows us to compute the number N. It suffices to set $N := \max\{h, \lceil \log_c(\varepsilon \cdot (1-c)) \rceil\}$. Based on the bounds on c and h, this allows us to conclude that $N \in \exp(||\mathcal{A}||^{O(1)})$, see [3]. In the rest of this section we prove Lemma 1.

We start with two preprocessing steps. First, we make sure that $T = \emptyset$, resulting in $\mathrm{Val}(\mathrm{Term}, (q,i)) = \sup_\sigma \mathbb{P}^\sigma_{(q,i)}(\mathrm{Term} \cap \text{not reach } T)$. Second, we make sure that there are no "degenerate" states in the system which would enable a strategy to spend an unbounded time with a bounded positive counter value. Both these reductions will be carried out in deterministic polynomial time.

In more detail, the first reduction step takes $\mathcal{A}$ and outputs $\mathcal{A}'$ given by replacing T with a single fresh control state, q_D ("D" for "diverging"), equipped with a single outgoing rule $(q_D, +1, q_D)$. By results of [4], this can be done in polynomial time. Obviously, for $q \notin T$ the value $\sup_\sigma \mathbb{P}^\sigma_{(q,i)}(\mathrm{Term} \cap \text{not reach } T)$ is the same in both $\mathcal{A}^\infty$ and $\mathcal{A}'^\infty$. Thus we may assume that $T = \emptyset$ when proving Lemma 1.

In the second reduction step, the property we need to assure holds in $\mathcal{A}$ is best stated in terms of $\mathcal{G}_{\mathcal{A}}$ and the variables $R^{(i)}$. We need to guarantee that under every pure memoryless strategy, $\liminf_{i\to\infty} R^{(i)}/i$ is almost surely positive.[2]

[2] This value is sometimes called the mean payoff, see also [4].

$$
\begin{aligned}
&z_q \le -x + k + z_r && \text{for all } q \in Q_1 \text{ and } (q,k,r) \in \delta,\\
&z_q \le -x + \textstyle\sum_{(q,k,r)\in\delta} P_q((q,k,r)) \cdot (k + z_r) && \text{for all } q \in Q_0,\\
&x > 0.
\end{aligned}
$$

Fig. 1. The system $\mathcal{L}$ of linear inequalities over x and z_q, $q \in Q$.

For runs starting in a state q, we denote by V_q the random variable giving the first time when q is revisited, or ∞ if it is not revisited. Let us call a pure memoryless strategy, σ, for $\mathcal{G}_{\mathcal{A}}$ *idling* if there is a state, q, such that $\mathbb{P}_q^{\sigma}\left(V_q < \infty\right) = 1$ and $\mathbb{P}_q^{\sigma}\left(R^{(V_q)} = 0\right) = 1$. We want to modify the OC-MDP, so that idling is not possible, without influencing the termination value. A technique to achieve this was already developed in our previous work [4], where we used the term "decreasing" for non-idling strategies. There we gave a construction which preserves the property of optimal termination probability being $= 1$. We in fact can establish that that construction preserves the exact termination value. Because the idea is not new, we leave details to [3].

After performing both reduction steps, we can safely assume that $T = \emptyset$ and that there are no idling pure memoryless strategies. The next claim then follows from Lemma 10 in [4]:

Claim. Under the assumptions above, for every pure memoryless strategy, σ, for $\mathcal{G}_{\mathcal{A}}$, and every $q \in Q$ we have $\mathbb{P}_q^{\sigma}\left(\liminf_{i\to\infty} R^{(i)}/i > 0\right) = 1$.

We shall now introduce a linear system of inequalities, $\mathcal{L}$, which is closely related to a standard LP that one can associate with a finite-state MDP with rewards, in order to obtain its optimal mean payoff value. The solution to the system of inequalities $\mathcal{L}$ will allow us to define a (sub)martingale that is critical for our arguments. This is an extension, to OC-MDPs, of a method used in [7] for analysis of purely probabilistic one-counter machines. The variables of $\mathcal{L}$ are x and z_q, for all $q \in Q$. The linear inequalities are defined in Figure 1.

Lemma 2. *There is a non-negative rational solution* $(\bar{x}, (\bar{z}_q)_{q\in Q}) \in \mathbb{Q}^{|Q|+1}$ *to* $\mathcal{L}$*, such that* $\bar{x} > 0$*. (The binary encoding size of the solution is polynomial in* $||\mathcal{A}||$*.)*

Proof. We first prove that there is some non-negative solution to $\mathcal{L}$ with $\bar{x} > 0$. The bound on size then follows by standard facts about linear programming. To find a solution, we will use optimal values of the MDP under the objective of minimizing *discounted total reward*. For every discount factor, λ, $0 < \lambda < 1$, there is a pure memoryless strategy, σ_λ, for $\mathcal{G}_{\mathcal{A}}$ such that $e_q^{\lambda}(\tau) := \sum_{i\ge 0} \lambda^i \cdot \mathbb{E}_q^{\tau}\left[R^{(i+1)} - R^{(i)}\right]$ is minimized by setting $\tau := \sigma_\lambda$. We prove that there is some λ, such that setting $\bar{z}_q := e_q^{\lambda}(\sigma_\lambda)$ and

$$
\begin{aligned}
\bar{x} := \min\big(&\{k + e_r^{\lambda}(\sigma_\lambda) - e_q^{\lambda}(\sigma_\lambda) \mid q \in Q_1, (q,k,r) \in \delta\}\\
&\cup \{P_q((q,k,r)) \cdot \left(k + e_r^{\lambda}(\sigma_\lambda) - e_q^{\lambda}(\sigma_\lambda)\right) \mid q \in Q_0, (q,k,r) \in \delta\}\big)
\end{aligned}
$$

forms a non-negative solution to $\mathcal{L}$ with $\bar{x} > 0$.

Now we proceed in more detail. By standard results (e.g., [13]), for a fixed state, q, and a fixed discount, $\lambda < 1$, there is always a pure memoryless strategy, σ_q, minimizing

$e_q^\lambda(\tau)$ in place of τ. As we already proved in the Claim above, due to our assumptions we have $\mathbb{P}_q^{\sigma_q}\left(\liminf_{i\to\infty} R^{(i)}/i > 0\right) = 1$. Thus $\sum_{i\geq 0} \cdot \mathbb{E}_q^\tau\left[R^{(i+1)} - R^{(i)}\right] = \infty$, and there is a $\lambda < 1$ such that $e_q^\lambda(\sigma_q) > 0$ for all $q \in Q$. Finally, observe that there is a single strategy, σ_λ, which can be used as σ_q for all q. This is a consequence of σ_q being optimal also in successors of q. Finally, $\bar{x} > 0$, because for all $q \in Q_0$

$$\begin{aligned} e_q^\lambda(\sigma_\lambda) &= \sum_{i\geq 0} \lambda^i \cdot \mathbb{E}_q^{\sigma_\lambda}\left[R^{(i+1)} - R^{(i)}\right] \\ &= \sum_{(q,k,r)\in\delta} P_q((q,k,r)) \cdot \left(k + \lambda \cdot \sum_{i\geq 0} \lambda^i \cdot \mathbb{E}_r^{\sigma_\lambda}\left[R^{(i+1)} - R^{(i)}\right]\right) \\ &= \sum_{(q,k,r)\in\delta} P_q((q,k,r)) \cdot \left(k + \lambda \cdot e_r^\lambda(\sigma_\lambda)\right) \\ &< \sum_{(q,k,r)\in\delta} P_q((q,k,r)) \cdot \left(k + e_r^\lambda(\sigma_\lambda)\right), \end{aligned}$$

the last inequality following from $e_r^\lambda(\sigma_\lambda) > 0$ for all $r \in Q$; and similarly for all $q \in Q_1$ and $(q,k,r) \in \delta$

$$\begin{aligned} e_q^\lambda(\sigma_\lambda) &= \sum_{i\geq 0} \lambda^i \cdot \mathbb{E}_q^{\sigma_\lambda}\left[R^{(i+1)} - R^{(i)}\right] \\ &\leq k + \lambda \cdot \sum_{i\geq 0} \lambda^i \cdot \mathbb{E}_r^{\sigma_\lambda}\left[R^{(i+1)} - R^{(i)}\right] = k + \lambda \cdot e_r^\lambda(\sigma_\lambda) < k + e_r^\lambda(\sigma_\lambda). \end{aligned}$$

□

Recall the random variables $\{C^{(i)}\}_{i\geq 0}$ and $\{S^{(i)}\}_{i\geq 0}$, returning the height of the counter, and the control state after completing i transitions. Given the solution $(\bar{x}, (\bar{z}_q)_{q\in Q}) \in \mathbb{Q}^{|Q|+1}$ from Lemma 2, we define a sequence of random variables $\{m^{(i)}\}_{i\geq 0}$ by setting

$$m^{(i)} := \begin{cases} C^{(i)} + \bar{z}_{S^{(i)}} - i \cdot \bar{x} & \text{if } C^{(j)} > 0 \text{ for all } j,\ 0 \leq j < i, \\ m^{(i-1)} & \text{otherwise.} \end{cases}$$

We shall now show that $m^{(i)}$ defines a submartingale. For relevant definitions of (sub)martingales see, e.g., [11].

Lemma 3. *Under an arbitrary strategy, τ, for $\mathcal{A}^\infty$, and with an arbitrary initial configuration (q,n), the process $\{m^{(i)}\}_{i\geq 0}$ is a submartingale.*

Proof. Consider a fixed path, u, of length $i \geq 0$. For all j, $0 \leq j \leq i$ the values $C^{(j)}(\omega)$ are the same for all $\omega \in Run(u)$. We denote these common values by $C^{(j)}(u)$, and similarly for $S^{(j)}(u)$ and $m^{(j)}(u)$. If $C^{(j)}(u) = 0$ for some $j \leq i$, then $m^{(i+1)}(\omega) = m^{(i)}(\omega)$ for every $\omega \in Run(u)$. Thus $\mathbb{E}_{(q,n)}^\tau\left[m^{(i+1)} \mid Run(u)\right] = m^{(i)}(u)$. Otherwise, consider the last configuration, (r,l), of u. For every possible successor, (r',l'), set

$$p_{(r',l')} := \begin{cases} \tau(u)((r,l) \to (r',l')) & \text{if } r \in Q_1, \\ Prob((r,l) \to (r',l')) & \text{if } r \in Q_0. \end{cases}$$

Then

$$\mathbb{E}^{\tau}_{(q,n)}\Big[C^{(i+1)} - C^{(i)} + \bar{z}_{S^{(i+1)}} - \bar{x} \mid \mathit{Run}(u)\Big] = \quad -\bar{x} + \sum_{(r,k,r')\in\delta} p_{(r',l+k)} \cdot (k + \bar{z}_{r'}) \quad \geq \quad \bar{z}_r.$$

This allows us to derive the following:

$$\begin{aligned}\mathbb{E}^{\tau}_{(q,n)}\Big[m^{(i+1)} \mid \mathit{Run}(u)\Big] &= \mathbb{E}^{\tau}_{(q,n)}\Big[C^{(i+1)} + \bar{z}_{S^{(i+1)}} - (i+1)\cdot\bar{x} \mid \mathit{Run}(u)\Big] \\ &= C^{(i)}(u) + \mathbb{E}^{\tau}_{(q,n)}\Big[C^{(i+1)} - C^{(i)} + \bar{z}_{S^{(i+1)}} - \bar{x} \mid \mathit{Run}(u)\Big] - i\cdot\bar{x} \\ &\geq C^{(i)}(u) + \bar{z}_{S^{(i)}(u)} - i\cdot\bar{x} \quad = \quad m^{(i)}(u). \end{aligned}$$

□

Now we can finally prove Lemma 1. Denote by Term_j the event of terminating after *exactly* j steps. Further set $\bar{z}_{\max} := \max_{q\in Q} \bar{z}_q - \min_{q\in Q} \bar{z}_q$, and assume that $C^{(0)} \geq \bar{z}_{\max}$. Then the event Term_j implies that $m^{(j)} - m^{(0)} = \bar{z}_{S^{(j)}} - j\cdot\bar{x} - C^{(0)} - \bar{z}_{S^{(0)}} \leq -j\cdot\bar{x}$. Finally, observe that we can bound the one-step change of the submartingale value by $\bar{z}_{\max} + \bar{x} + 1$. Using the Azuma-Hoeffding inequality for the submartingale $\{m^{(n)}\}_{n\geq 0}$ (see, e.g., Theorem 12.2.3 in [11]), we thus obtain the following bound for every strategy σ and initial configuration (q,i) with $i \geq \bar{z}_{\max}$:

$$\mathbb{P}^{\sigma}_{(q,i)}\Big(\mathrm{Term}_j\Big) \leq \mathbb{P}^{\sigma}_{(q,i)}\Big(m^{(j)} - m^{(0)} \leq -j\cdot\bar{x}\Big) \leq \exp\left(\frac{-\bar{x}^2\cdot j^2}{2j\cdot(\bar{z}_{\max}+\bar{x}+1)}\right).$$

We choose $c := \exp\left(\frac{-\bar{x}^2}{2\cdot(\bar{z}_{\max}+\bar{x}+1)}\right) < 1$ and observe that

$$\mathbb{P}^{\sigma}_{(q,i)}(\mathrm{Term}) = \sum_{j\geq i} \mathbb{P}^{\sigma}_{(q,i)}\Big(\mathrm{Term}_j\Big) \leq \sum_{j\geq i} c^j = \frac{c^i}{1-c}.$$

This choice of c, together with $h := \lceil \bar{z}_{\max} \rceil$, finishes the proof of Lemma 1. (The given bounds on c and h are easy to check.) □

3.3 Bounding N for General SSGs

For a control state q, let $\nu_q := \sup_\sigma \inf_\pi \mathbb{P}^{\sigma,\pi}_q(\mathit{LimInf}(=-\infty))$. Given a OC-SSG, $\mathcal{A} = (Q, (Q_0, Q_1, Q_2), \delta, P)$, and $\varepsilon > 0$, we now show how to obtain a computable bound on the number N such that $\big|\mathrm{Val}(\mathrm{Term}, (q,i)) - \nu_q\big| < \varepsilon$ for all $i \geq N$. Again, by the arguments described in Section 3, once we have this, we have an algorithm for approximating $\mathrm{Val}(\mathrm{Term}, (q,i))$.

By results in [2], there is always a counterless pure strategy, π^*, for Player 2 in $\mathcal{G}_{\mathcal{A}}$, such that

$$\sup_\sigma \inf_\pi \mathbb{P}^{\sigma,\pi}_q(\mathit{LimInf}(=-\infty)) = \sup_\sigma \mathbb{P}^{\sigma,\pi^*}_q(\mathit{LimInf}(=-\infty)).$$

Observe that by fixing the choices of π^* in $\mathcal{A}$ we obtain a maximizing OC-MDP, $\mathcal{A}^* = (Q^*, (Q^*_0, Q^*_1), \delta^*, P^*)$, where $Q^*_0 = Q_0 \cup Q_2$, $Q^*_1 = Q_1$, $\delta^* := \{(q,k,r) \in \delta \mid q \in Q_0 \cup Q_1 \vee \pi^*(q) = r\}$, and P^* is the unique (for $\mathcal{A}^*$) extension of P to states from Q_2.

Slightly abusing notation, denote also by π^* the strategy for $\mathcal{A}^\infty$ which corresponds, in the sense explained in Section 2, to π^* for $\mathcal{G}_{\mathcal{A}}$. Then $\nu_q = \mathbb{P}^{\sigma,\pi^*}_q(\mathit{LimInf}(=-\infty)) \leq \mathrm{Val}(\mathrm{Term}, (q,i)) \leq \mathbb{P}^{\sigma,\pi^*}_{(q,i)}(\mathrm{Term})$. Applying Lemma 1 to the OC-MDP $\mathcal{A}^*$ thus allows us to give a computable (exponential) bound on N, given $\mathcal{A}$.

References

1. Berger, N., Kapur, N., Schulman, L.J., Vazirani, V.: Solvency Games. In: Proc. of FSTTCS 2008 (2008)
2. Brázdil, T., Brožek, V., Etessami, K.: One-Counter Simple Stochastic Games. In: Proc. of FSTTCS 2010, pp. 108–119 (2010)
3. Brázdil, T., Brožek, V., Etessami, K., Kučera, A.: Approximating the Termination Value of One-Counter MDPs and Stochastic Games. Tech. Rep. abs/1104.4978, CoRR (2011), http://arxiv.org/abs/1104.4978
4. Brázdil, T., Brožek, V., Etessami, K., Kučera, A., Wojtczak, D.: One-Counter Markov Decision Processes. In: ACM-SIAM SODA. pp. 863–874 (2010), full tech. report: CoRR, abs/0904.2511 (2009), http://arxiv.org/abs/0904.2511
5. Brázdil, T., Brožek, V., Forejt, V., Kučera, A.: Reachability in recursive Markov decision processes. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 358–374. Springer, Heidelberg (2006)
6. Brázdil, T., Brožek, V., Kučera, A., Obdržálek, J.: Qualitative Reachability in stochastic BPA games. In: Proc. 26th STACS, pp. 207–218 (2009)
7. Brázdil, T., Kiefer, S., Kučera, A.: Efficient analysis of probabilistic programs with an unbounded counter. CoRR abs/1102.2529 (2011)
8. Etessami, K., Wojtczak, D., Yannakakis, M.: 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), pp. 243–253 (2008)
9. Etessami, K., Yannakakis, M.: Recursive Markov decision processes and recursive stochastic games. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 891–903. Springer, Heidelberg (2005)
10. Etessami, K., Yannakakis, M.: Efficient qualitative analysis of classes of recursive Markov decision processes and simple stochastic games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, Springer, Heidelberg (2006)
11. Grimmett, G.R., Stirzaker, D.R.: Probability and Random Processes, 2nd edn. Oxford U. Press, Oxford (1992)
12. Lambert, J., Van Houdt, B., Blondia, C.: A policy iteration algorithm for markov decision processes skip-free in one direction. In: ValueTools. ICST, Brussels (2007)
13. Puterman, M.L.: Markov Decision Processes. J. Wiley and Sons, Chichester (1994)
14. White, L.B.: A new policy iteration algorithm for Markov decision processes with quasi birth-death structure. Stochastic Models 21, 785–797 (2005)

Generic Expression Hardness Results for Primitive Positive Formula Comparison

Simone Bova[1], Hubie Chen[2], and Matthew Valeriote[3]

[1] Department of Mathematics
Vanderbilt University, Nashville, USA
simone.bova@vanderbilt.edu
[2] Departament de Tecnologies de la Informació i les Comunicacions
Universitat Pompeu Fabra, Barcelona, Spain
hubie.chen@upf.edu
[3] Department of Mathematics & Statistics
McMaster University, Hamilton, Canada
matt@math.mcmaster.ca

Abstract. We study the expression complexity of two basic problems involving the comparison of primitive positive formulas: equivalence and containment. We give two generic hardness results for the studied problems, and discuss evidence that they are optimal and yield, for each of the problems, a complexity trichotomy.

1 Introduction

Overview. A *primitive positive (pp)* formula is a first-order formula defined from atomic formulas and equality of variables using conjunction and existential quantification. The class of primitive positive formulas includes, and is essentially equivalent to, the class of *conjunctive queries*, which is well-established in relational database theory as a pertinent and useful class of queries, and which has been studied complexity-theoretically from a number of perspectives (see for example [18,17,1]). In this paper, we study the complexity of the following fundamental problems, each of which involves the comparison of two pp-formulas ϕ, ϕ' having the same free variables, over a relational structure.

- Equivalence: are the formulas ϕ, ϕ' equivalent–that is, do they have the same satisfying assignments–over the structure?
- Containment: are the satisfying assignments of ϕ contained in those of ϕ', over the structure?

We study the complexity of these computational problems with respect to various fixed structures. That is, we parameterize each of these problems with respect to the structure to obtain a family of problems, containing one member for each structure, and study the resulting families of problems. To employ the terminology of Vardi [19], we study the *expression complexity* of the presented comparison tasks. The suggestion here is that various relational structures–which

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 344–355, 2011.

may represent databases or knowledge bases, according to use–may possess structural characteristics that affect the complexity of the resulting problems, and our interest is in understanding this interplay. The present work focuses on relational structures that are finite (that is, have finite universe), and we assume that the structures under discussion are finite.

In this paper, we present two general expression hardness results on the problems of interest. In particular, each of our two main results provides a sufficient condition on a structure so that the problems are hard for certain complexity classes. Furthermore, we give evidence that our results are optimal, in that the conditions that they involve in fact describe dichotomies in the complexity of the studied problems; put together, our results indicate, for each of the studied problems, a complexity trichotomy.

Our study utilizes universal-algebraic tools that aid in understanding the set of primitive positive relations over a given structure. It is known that, relative to a structure, the set of relations that are definable by a primitive positive formula forms a robust algebraic object known as a *relational clone*; a known Galois correspondence associates, in a bijective manner, each such relational clone with a *clone*, a set of operations with certain closure properties. This correspondence provides a way to pass from a relational structure $\mathbf{B}$ to an algebra $\mathbb{A}_{\mathbf{B}}$ whose set of operations is the mentioned clone, in such a way that two structures having the same algebra have the same complexity (for each of the mentioned problems). In a previous paper by the present authors [6], we developed this correspondence and presented some basic complexity results for the problems at hand, including a classification of the complexity of the problems on all two-element structures.

Hardness Results. Our first hardness result yields that for any structure $\mathbf{B}$ whose associated algebra $\mathbb{A}_{\mathbf{B}}$ gives rise to a variety $\mathcal{V}(\mathbb{A}_{\mathbf{B}})$ that *admits the unary type*, both the equivalence and containment problems are Π_2^p-complete. Note that this is the maximal complexity possible for these problems, as the problems are contained in the class Π_2^p. The condition of admitting the unary type originates from tame congruence theory, a theory developed to understand the structure of finite algebras [13]. We observe that this result implies a dichotomy in the complexity of the studied problems under the *G-set conjecture* for the constraint satisfaction problem (CSP), a conjecture put forth by Bulatov, Jeavons, and Krokhin [7] which predicts exactly where the tractability/intractability dichotomy lies for the CSP. In particular, under the G-set conjecture, the structures not obeying the described condition have equivalence and containment problems in coNP. The resolution of the G-set conjecture, on which there has been focused and steady progress over the past decade [9,14,10,3], would thus, in combination with our hardness result, yield a coNP/Π_2^p-complete dichotomy for the equivalence and containment problems. In fact, our hardness result already unconditionally implies dichotomies for our problems for all classes of structures where the G-set conjecture has already been established, including the class of three-element structures [9], and the class of conservative structures [8].

One formulation of the G-set conjecture is that, for a structure $\mathbf{B}$ whose associated algebra $\mathbb{A}_{\mathbf{B}}$ is idempotent, the absence of the unary type in the variety

generated by $\mathbb{A}_{\mathbf{B}}$ implies that $\mathsf{CSP}(\mathbf{B})$ is polynomial-time tractable. The presence of the unary type is a known sufficient condition for intractability in the idempotent case [7,10], and this conjecture predicts exactly where the tractability/intractability dichotomy lies for the CSP. It should be noted, however, that the boundary that is suggested by our hardness result for the equivalence and containment problems is *not* the same as the boundary suggested by the G-set conjecture for the CSP. The G-set conjecture, which is typically phrased on idempotent algebras, yields a prediction on the CSP complexity of all structures via a theorem [7] showing that each structure $\mathbf{B}$ has the same CSP complexity as a structure $\mathbf{B}'$ whose associated algebra is idempotent. The mapping from $\mathbf{B}$ to $\mathbf{B}'$ does not preserve the complexity of the problems studied here, and indeed, there are examples of two-element structures $\mathbf{B}$ such that our hardness result applies to $\mathbf{B}$–the equivalence and containment problems on $\mathbf{B}$ are Π_2^p-complete–but $\mathbf{B}'$ does not admit the unary type and indeed has a polynomial-time tractable CSP [6]. Our new result requires establishing a deeper understanding of the identified algebras' structure, some of which admit a tractable CSP, in order to obtain hardness.

Our second hardness result shows that for any structure $\mathbf{B}$, if the variety $\mathcal{V}(\mathbb{A}_{\mathbf{B}})$ is not congruence modular, then the equivalence and containment problems are coNP-hard. Previous work identified one most general condition for the tractability of the equivalence and containment problems: if the algebra *has few subpowers*–a combinatorial condition [4,14] involving the number of subalgebras of powers of an algebra–then these problems are polynomial-time tractable [6, Theorem 7]. This second hardness result appears to perfectly complement this tractability result: there are no known examples of algebras $\mathbb{A}_{\mathbf{B}}$ (of structures $\mathbf{B}$ having finitely many relations) that are not covered by one of these results, and in fact the *Edinburgh conjecture* predicts that none exist, stating that every such algebra $\mathbb{A}_{\mathbf{B}}$ that generates a congruence modular variety also has few subpowers. Concerning this conjecture, it should be pointed out that the resolution of the *Zadóri conjecture*, a closely related conjecture of which the Edinburgh conjecture is a generalization, was recently announced by Libor Barto [2]. The Edinburgh conjecture is of current interest, with recent work presented by Ralph McKenzie and colleagues. We also point out that this conjecture (as with the Zadóri conjecture) is purely algebraic, making no references to notions of computation.

In summary, up to polynomial-time computation, *we completely resolve the complexity of the studied problems on all finite structures, showing a P/coNP-complete/Π_2^p-complete trichotomy modulo two conjectures;* one is computational and one is algebraic, and for each there is both highly non-trivial supporting evidence and current investigation. This trichotomy follows from Theorems 4, 5, 6, and 7.

2 Preliminaries

Here, a *signature* is a set of relation symbols, each having an associated arity; we assume that all signatures are of finite size. A *relational structure* over a

signature σ consists of a universe B and, for each relation symbol $R \in \sigma$, a relation $R^{\mathbf{B}} \subseteq B^k$ where k is the arity of R. We assume that all relational structures under discussion have universes of finite size. A *primitive positive formula (pp-formula)* on σ is a first-order formula formed using equalities on variables ($x = x'$), atomic formulas $R(x_1, \ldots, x_k)$ over σ, conjunction ($\wedge$), and existential quantification ($\exists$).

We now define the problems that will be studied.

Definition 1. *We define the following computational problems; in each, an instance consists of a relational structure* $\mathbf{B}$ *and a pair* (ϕ, ϕ') *of pp-formulas over the signature of* $\mathbf{B}$ *having the same set of free variables* X*.*

- PPEQ*: decide if* ϕ *and* ϕ' *are equivalent, that is, whether for all* $f : X \to B$*, it holds that* $\mathbf{B}, f \models \phi$ *iff* $\mathbf{B}, f \models \phi'$*.*
- PPCON*: decide if* ϕ *is contained in* ϕ'*, that is, whether for all* $f : X \to B$*, it holds that* $\mathbf{B}, f \models \phi$ *implies* $\mathbf{B}, f \models \phi'$*.*

For every relational structure $\mathbf{B}$*, we define* $\mathsf{PPEQ}(\mathbf{B})$ *to be the problem* PPEQ *where the structure is fixed to be* $\mathbf{B}$*; hence, an instance of* $\mathsf{PPEQ}(\mathbf{B})$ *is just a pair* (ϕ, ϕ') *of pp-formulas. We define the family of problems* $\mathsf{PPCON}(\mathbf{B})$ *similarly.*

We now identify some basic complexity properties of these problems. First, the PPEQ and PPCON problems are contained in Π_2^p; this is straightforward to verify. Next, there is a direct reduction from $\mathsf{PPCON}(\mathbf{B})$ to $\mathsf{PPEQ}(\mathbf{B})$. Throughout the paper, the notion of reduction used is polynomial-time many-one reducibility.

Proposition 1. *For each structure* $\mathbf{B}$*, the problem* $\mathsf{PPCON}(\mathbf{B})$ *reduces to the problem* $\mathsf{PPEQ}(\mathbf{B})$*.*

We now review the relevant algebraic concepts to be used. An *algebra* is a pair $\mathbb{A} = (A, F)$ such that A is a nonempty set, called the *domain* or *universe* of the algebra, and F is a set of finitary operations on A.

Let B be a nonempty set, let f be an n-ary operation on B, and let R be a k-ary relation on B. We say that f *preserves* R (or f is a *polymorphism* of R), if for every n tuples $t_1, \ldots, t_n \in R$, denoting the tuple t_i by $(t_{i,1}, \ldots, t_{i,k})$, it holds that the tuple $f(t_1, \ldots, t_n) = (f(t_{1,1}, \ldots, t_{n,1}), \ldots, f(t_{1,k}, \ldots, t_{n,k}))$ is in R. We say that a relation R is *compatible* with a set of operations if it is preserved by all of the operations. We extend this terminology to relational structures: an operation f is a polymorphism of a relational structure $\mathbf{B}$ if f is a polymorphism of every relation of $\mathbf{B}$. We use $\mathsf{Pol}(\mathbf{B})$ to denote the set of all polymorphisms of a relational structure $\mathbf{B}$, and use $\mathbb{A}_{\mathbf{B}}$ to denote the algebra $(B, \mathsf{Pol}(\mathbf{B}))$. Dually, for an operation f, we use $\mathsf{Inv}(f)$ to denote the set of all relations that are preserved by f, and for a set of operations F, we define $\mathsf{Inv}(F)$ as $\bigcap_{f \in F} \mathsf{Inv}(f)$. We will make use of the following result connecting the $\mathsf{Pol}(\cdot)$ and $\mathsf{Inv}(\cdot)$ operators to pp-definability.

Theorem 1. *(Geiger [12], Bodcharnuk et al. [5]) Let* $\mathbf{B}$ *be a finite relational structure. The set of relations* $\mathsf{Inv}(\mathsf{Pol}(\mathbf{B}))$ *is equal to the set of relations that are pp-definable over* $\mathbf{B}$*.*

We associate to each algebra $\mathbb{A} = (A, F)$ a set of problems $\mathsf{PPEQ}(\mathbb{A})$, namely, the set containing all problems $\mathsf{PPEQ}(\mathbf{B})$ where $\mathbf{B}$ has universe A and $F \subseteq \mathsf{Pol}(\mathbf{B})$. We define $\mathsf{PPCON}(\mathbb{A})$ similarly. For a complexity class $\mathcal{C}$, we say that $\mathsf{PPEQ}(\mathbb{A})$ is $\mathcal{C}$-hard if $\mathsf{PPEQ}(\mathbb{A})$ contains a problem $\mathsf{PPEQ}(\mathbf{B})$ that is $\mathcal{C}$-hard. We define $\mathcal{C}$-hardness similarly for $\mathsf{PPCON}(\mathbb{A})$.

Theorem 2. *Let* $\mathbf{B}$ *be a finite relational structure, and let* $\mathcal{C}$ *be a complexity class closed under polynomial-time many-one reductions. The problem* $\mathsf{PPEQ}(\mathbf{B})$ *is* $\mathcal{C}$*-hard if and only if* $\mathsf{PPEQ}(\mathbb{A}_{\mathbf{B}})$ *is* $\mathcal{C}$*-hard. The same result holds for* $\mathsf{PPCON}(\cdot)$.

The notion of a *variety* is typically defined on indexed algebras; a variety is a class of similar algebras that is closed under the formation of homomorphic images, subalgebras, and products. For our purposes here, however, we may note that the variety generated by an algebra $\mathbb{A}$, denoted by $\mathcal{V}(\mathbb{A})$, is known to be equal to $HSP(\{\mathbb{A}\})$, where the operator H (for instance) is the set of algebras derivable by taking homomorphic images of algebras in the given argument set.

Theorem 3. *Suppose that* $\mathbb{B} \in \mathcal{V}(\mathbb{A})$. *Then, for every problem* $\mathsf{PPEQ}(\mathbf{B}) \in \mathsf{PPEQ}(\mathbb{B})$, *there exists a problem* $\mathsf{PPEQ}(\mathbf{B}') \in \mathsf{PPEQ}(\mathbb{A})$ *such that* $\mathsf{PPEQ}(\mathbf{B})$ *reduces to* $\mathsf{PPEQ}(\mathbf{B}')$, *and likewise for* $\mathsf{PPCON}(\cdot)$.

3 Unary Type

In this section, we present the first hardness result described in the introduction.

Our proof makes use of the detailed information on tame congruence theory provided in [13] and [16]. This theory associates a *typeset* to a non-trivial finite algebra, which contains one or more of five *types*: (1) the unary type, (2) the affine type, (3) the boolean type, (4) the lattice type, and (5) the semilattice type. By extension, a typeset is associated to each variety, namely, the union of all typesets of finite algebras contained in the variety. A variety is said to *admit* a type if the type is contained in its typeset, and is otherwise said to *omit* the type.

Theorem 4. *Let* $\mathbf{B}$ *be a finite relational structure. If* $\mathcal{V}(\mathbb{A}_{\mathbf{B}})$ *admits the unary type, then* $\mathsf{PPEQ}(\mathbf{B})$ *and* $\mathsf{PPCON}(\mathbf{B})$ *are* Π_2^p*-hard.*

As we now show, the previous theorem implies, modulo the G-Set conjecture, a coNP/Π_2^p-complete dichotomy for the equivalence and containment problems.

Theorem 5. *Let* $\mathbf{B}$ *be a finite relational structure over a finite signature. If the G-Set conjecture holds and* $\mathcal{V}(\mathbb{A}_{\mathbf{B}})$ *omits the unary type, then both* $\mathsf{PPEQ}(\mathbf{B})$ *and* $\mathsf{PPCON}(\mathbf{B})$ *are contained in coNP.*

Proof. Let $\mathbf{B}^*$ be obtained from $\mathbf{B}$ by adding to it all relations of the form $\{b\}$ for $b \in B$. If $\mathbf{B}$ is a finite relational structure such that the variety generated by $\mathbb{A}_{\mathbf{B}}$ omits the unary type, then the variety generated by $\mathbb{A}_{\mathbb{B}^*}$ omits the unary type also, and according to the G-Set conjecture, $\mathsf{CSP}(\mathbf{B}^*)$ is in P. But then, $\mathsf{CSP}(\mathbf{B})$ is in P, since it can be viewed as a case of $\mathsf{CSP}(\mathbf{B}^*)$. From this it follows that $\mathsf{PPEQ}(\mathbf{B})$ and $\mathsf{PPCON}(\mathbf{B})$ are in coNP. □

In order to prove Theorem 4, we will first establish the key algebraic lemma (Lemma 1). Then, we will establish the desired hardness result (Theorem 4), by reducing from the containment problem over a boolean structure having constant polymorphisms, known to be Π_2^p-complete [6], to the containment problem of interest.

3.1 Algebra

We prepare the key algebraic lemma, whose proof requires a certain amount of the theory of tame congruences and multitraces [13,16].

Lemma 1. *Let* $\mathbf{B}$ *be a finite relational structure such that* $\mathcal{V}(\mathbb{A}_\mathbf{B})$ *admits the unary type, and let* $\mathbf{C} = (\{0,1\}, \{C_1, \ldots, C_l\})$ *be a relational structure whose relations contain the constant tuples. Then, there is a finite algebra* $\mathbb{A} \in \mathcal{V}(\mathbb{A}_\mathbf{B})$ *and a finite set* $\mathcal{R} = \{D_1, \ldots, D_l\} \cup \{E_1, \ldots, E_k\}$ *of finitary relations over* A *(where* $k = |A|$*), compatible with the operations of* $\mathbb{A}$*, satisfying the following.*

Let $\mathbf{A} = (A, \mathcal{R})$*. Let* $\phi(x_1, \ldots, x_m)$ *be a pp-formula on* $\mathbf{C}$ *with quantified variables* $x_{m+1}, \ldots, x_n$*. Define the pp-formula* $\phi'(x_1, \ldots, x_m)$ *over* $\mathbf{A}$ *by replacing each atomic formula* $C_i(z_1, \ldots, z_r)$ *in* ϕ *by* $D_i(z_1, \ldots, z_r)$*, and conjoining* $E_n(x_1, \ldots, x_n)$ *if* $n \leq k$*, and* $\bigwedge_{1 \leq i_1 < \cdots < i_k \leq n} E_k(x_{i_1}, \ldots, x_{i_k})$*, otherwise. Then,* $(\phi, \psi) \in \mathsf{PPCON}(\mathbf{C})$ *if and only if* $(\phi', \psi') \in \mathsf{PPCON}(\mathbf{A})$*.*

3.2 Reduction

We now prove Theorem 4.

Proof (Theorem 4). By Theorem 4 and Lemma 4 from [6] it follows that there is some Boolean relational structure $\mathbf{C} = (\{0,1\}, \{C_1, \ldots, C_l\})$, whose relations contain the constant tuples, such that $\mathsf{PPCON}(\mathbf{C})$ is Π_2^p-complete. Let $\mathbf{A} = (A, \{D_1, \ldots, D_l, E_1, \ldots, E_k\})$ be the finite relational structure defined in terms of Lemma 1 over the universe A of the finite algebra $\mathbb{A} \in \mathcal{V}(\mathbb{A}_\mathbf{B})$. The construction of a pp-formula ϕ' over $\mathbf{A}$ from a pp-formula ϕ over $\mathbf{C}$ given in the lemma provides a reduction from $\mathsf{PPCON}(\mathbf{C})$ to $\mathsf{PPCON}(\mathbf{A})$. This reduction is polynomial-time since the number of relations E_j that are conjoined to ϕ' can be bounded by a polynomial in n. □

4 Non-congruence Modularity

We present the second hardness result described in the introduction. An algebra $\mathbb{A}$ is said to be congruence modular, if its lattice of congruences satisfies the modular law, $\forall x \forall y \forall z (x \leq y \rightarrow x \vee (y \wedge z) = y \wedge (x \vee z))$. A variety is said to be congruence modular if all of its members are congruence modular.

Theorem 6. *Let* $\mathbf{B}$ *be a finite relational structure. If* $\mathcal{V}(\mathbb{A}_\mathbf{B})$ *is not congruence modular, then* $\mathsf{PPEQ}(\mathbf{B})$ *and* $\mathsf{PPCON}(\mathbf{B})$ *are coNP-hard.*

The previous theorem implies, modulo the Edinburgh conjecture, a P/coNP-hard dichotomy for the equivalence and containment problems.

Theorem 7. *Let* $\mathbf{B}$ *be a finite relational structure over a finite signature. If the Edinburgh conjecture holds and* $\mathcal{V}(\mathbb{A}_{\mathbf{B}})$ *is congruence modular, then* $\mathsf{PPEQ}(\mathbf{B})$ *and* $\mathsf{PPCON}(\mathbf{B})$ *are in P.*

Proof. By the Edinburgh conjecture, the congruence modularity of $\mathcal{V}(\mathbb{A}_{\mathbf{B}})$ implies that $\mathbb{A}_{\mathbf{B}}$ has few subpowers. Then, we have from [6, Theorem 7] that $\mathsf{PPEQ}(\mathbf{B})$ and $\mathsf{PPCON}(\mathbf{B})$ are in P. □

In order to prove Theorem 6, we first establish the key algebraic lemma (Lemma 2). Next, we give a sequence of reductions to establish the desired hardness result: we first reduce from the problem of deciding whether a DNF is a tautology, a known coNP-complete problem, to a certain comparison problem over lattices; we then reduce to a certain entailment problem on (sorted) relational structures which we call "pentagons"; finally, we reduce from pentagon entailment to the containment problem (Theorem 6).

4.1 Algebra

Let A be a set. For binary relations θ, θ' on A, the relational product $\theta \circ \theta'$ is the binary relation defined by $\{(a,b) \mid (a,c) \in \theta \text{ and } (c,b) \in \theta' \text{ for some } c\}$. We use θ^k to denote the k-fold relational product of θ with itself. We let $\mathrm{Eq}(A)$ denote the complete lattice of equivalence relations on A, and we let $0_A = \{(a,a) \mid a \in A\}$ and $1_A = A^2$ denote the bottom and top elements of $\mathrm{Eq}(A)$, respectively.

Proposition 2. *Let* A *be a set such that* $|A| = m$. *Let* $\theta_1, \ldots, \theta_k \in \mathrm{Eq}(A)$. *It holds that* $\theta_1 \vee \cdots \vee \theta_k = (\theta_1 \circ \cdots \circ \theta_k)^m$.

A *pentagon* is a structure $\mathbf{P}$ over the signature $\{\alpha, \beta, \gamma\}$ containing three binary relation symbols such that $\alpha^{\mathbf{P}}$, $\beta^{\mathbf{P}}$, and $\gamma^{\mathbf{P}}$ are equivalence relations on P, and the following conditions hold in $\mathrm{Eq}(P)$: $\alpha^{\mathbf{P}} \leq \beta^{\mathbf{P}}$, $\beta^{\mathbf{P}} \wedge \gamma^{\mathbf{P}} = 0_P$, $\beta^{\mathbf{P}} \circ \gamma^{\mathbf{P}} = 1_P$, and $\alpha^{\mathbf{P}} \vee \gamma^{\mathbf{P}} = 1_P$. We remark that in the sequence of reductions that we give, we do not make explicit use of the last item in the definition of pentagon.

A pentagon $\mathbf{P} = (P, \alpha^{\mathbf{P}}, \beta^{\mathbf{P}}, \gamma^{\mathbf{P}})$ can be naturally decomposed as a direct product $P = B \times C$ in such a way that $\beta^{\mathbf{P}}$ and $\gamma^{\mathbf{P}}$ are the kernels of the projections of P onto B and C, respectively. Each element $b \in B$ induces, via $\alpha^{\mathbf{P}}$, an equivalence relation α_b on C, namely

$$\alpha_b = \{(c,c') \mid ((b,c),(b,c')) \in \alpha^{\mathbf{P}}\} \in \mathrm{Eq}(C). \tag{1}$$

In associating together two elements $b, b' \in B$ when $\alpha_b = \alpha_{b'}$, one naturally obtains a partition of B into $l \geq 1$ non-empty blocks $B_1, \ldots, B_l$ and equivalence relations $\alpha_1, \ldots, \alpha_l$ on C such that for all $b \in B$, it holds that $\alpha_i = \alpha_b$ if and only if $b \in B_i$. We say that a pentagon is *interesting* if the sequence $\alpha_1, \ldots, \alpha_l$ contains equivalence relations α_j and α_k such that $\alpha_j < \alpha_k$ holds in $\mathrm{Eq}(C)$; we say that a set of pentagons is *interesting* if it contains an interesting pentagon.

Let $\mathbf{B}$ be a finite relational structure. If $\mathcal{V} = \mathcal{V}(\mathbb{A}_{\mathbf{B}})$ is not congruence modular, then this can be witnessed in the congruence lattice of the 4-generated free

algebra in $\mathcal{V}$. More precisely, let $\mathbb{F}_4$ be the $\mathcal{V}$-free algebra freely generated by $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{d}$ and let α^*, β, and γ be the congruences of $\mathbb{F}_4$ generated by $\{(\mathbf{a},\mathbf{b})\}$, $\{(\mathbf{a},\mathbf{b}),(\mathbf{c},\mathbf{d})\}$, and $\{(\mathbf{a},\mathbf{c}),(\mathbf{b},\mathbf{d})\}$ respectively. If we set $\alpha = \alpha^* \vee (\beta \wedge \gamma)$ then it follows that $\mathcal{V}$ will fail to be congruence modular if and only if $\alpha < \beta$ and that in this case, the three congruences α, β, and γ provide a witness to the failure of the modular law in the congruence lattice of $\mathbb{F}_4$ [11].

By working over a suitable quotient of the algebra $\mathbb{F}_4$, we establish the following Lemma.

Lemma 2. *Let* $\mathbf{B}$ *be a finite relational structure such that* $\mathcal{V}(\mathbb{A}_\mathbf{B})$ *is not congruence modular. There is a finite algebra* $\mathbb{A} \in \mathcal{V}(\mathbb{A}_\mathbf{B})$ *having congruences* α, β, *and* γ *such that* $\alpha < \beta$, $\gamma \wedge \beta = 0_A$, *and* $\alpha \vee \gamma = \beta \vee \gamma$. *Furthermore, there exists a finite interesting set of pentagons* $\mathcal{P}$, *and a finite set* $\mathcal{D}$ *of finitary relations over* A *compatible with the operations of* $\mathbb{A}$ *such that:*

(*i*) *If* $\mathbf{P} = (P, \alpha^\mathbf{P}, \beta^\mathbf{P}, \gamma^\mathbf{P}) \in \mathcal{P}$, *then* $P \subseteq A$ *and* $\alpha^\mathbf{P} = \alpha \cap P^2$, $\beta^\mathbf{P} = \beta \cap P^2$, *and* $\gamma^\mathbf{P} = \gamma \cap P^2$.

(*ii*) *For every* $k \geq 1$, *there exists a* k*-ary relation* D_k *on* A, *such that* D_k *has a pp-definition over the relations in* $\mathcal{D}$ *with size polynomial in* k, *and such that for all* $a_1, \ldots, a_k \in A$, $(a_1, \ldots, a_k) \in D_k$, *if and only if* $a_1, \ldots, a_k \in P$ *for some* $\mathbf{P} = (P, \alpha^\mathbf{P}, \beta^\mathbf{P}, \gamma^\mathbf{P}) \in \mathcal{P}$.

4.2 Reductions

From DNF-TAUTOLOGY **to Lattice Inequality.** A propositional formula is in disjunctive normal form (DNF) if it is a finitary disjunction ($\vee$) of finitary conjunctions ($\wedge$) of literals; a literal is a variable, x, or the negation of a variable, $\bar{x}$. The following problem is well-known to be coNP-complete.

Problem: DNF-TAUTOLOGY
Instance: A propositional formula ϕ in DNF.
Question: Is ϕ a tautology?

A *lattice term* t is an algebraic term over finitary joins and meets. The *depth* of t is the height of its syntactic tree. Let $L = (L, \wedge, \vee)$ be a lattice. We say that L is *nontrivial* if $|L| > 1$. Let $S \subseteq L$. Relative to a set of variables X, an S*-assignment* is a map $f\colon X \to S$.

For a set of lattices $\mathcal{L}$, we define the following computational problem.

Problem: DEPTH-4-TERM-INEQ($\mathcal{L}$)
Instance: A pair (t, t') of lattice terms of depth ≤ 4.
Question: Does $t \leq t'$ hold in all lattices $L \in \mathcal{L}$?

One has the following coNP-hardness result for this problem.

Theorem 8. *[15] Let* $\mathcal{L}$ *be a finite set of finite lattices containing a nontrivial lattice. The problem* DEPTH-4-TERM-INEQ($\mathcal{L}$) *is coNP-hard via a polynomial-time many-one reduction* f *from* DNF-TAUTOLOGY *satisfying the condition: if* (t, t') *is a no instance in the image of* f, *then for any lattice* $K \in \mathcal{L}$ *having elements* $a, a' \in K$ *with* $a < a'$, *there is an* $\{a, a'\}$*-assignment witnessing* $t \not\leq t'$ *in* K.

From Lattice Inequality to Pentagon Entailment. To each pentagon $\mathbf{P}$, we associate a 2-sorted structure $\mathbf{P}_2$ having B and C as first and second universe, respectively, where $P = B \times C$ according to the decomposition of P provided by the equivalence relations $\beta^{\mathbf{P}}$ and $\gamma^{\mathbf{P}}$. The structure $\mathbf{P}_2$ is over signature $\{R\}$ and has

$$R^{\mathbf{P}_2} = \{(b, c, c') \in B \times C \times C \mid b \in B_i \Rightarrow (c, c') \in \alpha_i\}. \tag{2}$$

We will be interested in sorted pp-formulas over the signature $\{R\}$. Such formulas are required to have a sort (1 or 2) associated with each variable; the permitted atomic formulas are equality between variables of the same sort and predicate applications having the form $R(x, y, y')$ where x has sort 1, and y and y' have sort 2. We define the following computational problem for each set $\mathcal{P}$ of pentagons.

Problem: 2-PENTAGON-ENTAILMENT($\mathcal{P}$)
Instance: A pair (ϕ, ψ) of sorted pp-formulas over the signature $\{R\}$ having the same free variables for each sort.
Question: Does $\phi \models \psi$ over all pentagons $\mathbf{P}_2$ with $\mathbf{P} \in \mathcal{P}$?

Theorem 9. *Let $\mathcal{P}$ be a finite set of finite pentagons containing an interesting pentagon. The problem* 2-PENTAGON-ENTAILMENT($\mathcal{P}$) *is coNP-hard via a polynomial-time reduction f from* DNF-TAUTOLOGY *satisfying the condition: if (ϕ, ψ) is a no instance in the image of f, then $\phi \not\models \psi$ is witnessed over $\mathbf{P}_2$ for any interesting pentagon $\mathbf{P} \in \mathcal{P}$.*

Proof. For a pentagon $\mathbf{P} \in \mathcal{P}$, let $P = B \times C$ be its decomposition, and let $\alpha_1, \ldots, \alpha_l$ be the equivalence relations on C associated to $\mathbf{P}$. Let $K_{\mathbf{P}}$ denote the sublattice of $\mathrm{Eq}(C)$ generated by $\alpha_1, \ldots, \alpha_l$. We define $\mathcal{L} = \{K_{\mathbf{P}} \mid \mathbf{P} \in \mathcal{P}\}$. Notice that $\mathcal{L}$ contains a nontrivial lattice, since there exists an interesting pentagon in $\mathcal{P}$. Hence, the problem DEPTH-4-TERM-INEQ($\mathcal{L}$) is coNP-hard by Theorem 8; let r denote the reduction given by this theorem.

Let $t(\mathbf{x})$ be a lattice term of depth less than or equal to 4 with $\mathbf{x} = (x_1, \ldots, x_n)$. By induction on the structure of t, we show how to construct a pp-formula $\phi_t(\mathbf{x}, y, y')$, where the variables of $\mathbf{x}$ are of sort 1 and the variables y, y' are of sort 2. This translation has the property (*): for all $b_1, \ldots, b_n \in B$ and for all $c, c' \in C$, $\phi_t(b_1, \ldots, b_n, c, c')$ holds in $\mathbf{P}_2$ if and only if (c, c') is in the equivalence relation given by $t^{K_{\mathbf{P}}}(\alpha_{b_1}, \ldots, \alpha_{b_n})$. If $t = x_i$, then $\phi_t(\mathbf{x}, y_1, y_2) = R(x_i, y_1, y_2)$. In this case, property (*) is straightforwardly verified from the definition of $R^{\mathbf{P}_2}$. If $t = t_1 \wedge \cdots \wedge t_k$, then $\phi_t(\mathbf{x}, y_1, y_2) = \phi_{t_1}(\mathbf{x}, y_1, y_2) \wedge \cdots \wedge \phi_{t_k}(\mathbf{x}, y_1, y_2)$. In this case, property (*) is straightforward to verify. If $t = t_1 \vee \cdots \vee t_k$, let $m = \max\{|C| \mid \mathbf{P} \in \mathcal{P}, P = B \times C\}$. Let $z_{0,k}$ and $z_{i,1}, \ldots, z_{i,k}$ for $i = 1, \ldots, m$ be variables such that $z_{0,k} = y_1$, $z_{m,k} = y_2$, and $z_{i,j}$ is a fresh variable of sort 2 otherwise. Then, $\phi_t(\mathbf{x}, y_1, y_2)$ is the pp-formula obtained by existentially quantifying the fresh variables $z_{i,j}$ before the conjunction

$$\bigwedge_{i=1}^{m} \left(\phi_{t_1}(\mathbf{x}, z_{i-1,k}, z_{i,1}) \wedge \bigwedge_{j=2}^{k} \left(\phi_{t_j}(\mathbf{x}, z_{i,j-1}, z_{i,j}) \right) \right).$$

In this case, property (*) follows from Proposition 2.

The desired reduction is the composition of the reduction r given by Theorem 8 with the mapping $(t, t') \to (\phi_t, \phi_{t'})$. We verify that the reduction is correct. Suppose that (t, t') is a yes instance in the image of r. Then, $t \leq t'$ holds in all lattices $K_{\mathbf{P}}$ with $\mathbf{P} \in \mathcal{P}$. It follows immediately from property (*) that $\phi_t \models \phi_{t'}$ over all pentagons $\mathbf{P}_2$ with $\mathbf{P} \in \mathcal{P}$. Suppose now that $(t(x_1, \ldots, x_n), t'(x_1, \ldots, x_n))$ is a no instance in the image of r. Let $\mathbf{P} \in \mathcal{P}$ be an interesting pentagon. There exist $b_1, b_2 \in B$ with $\alpha_{b_1} < \alpha_{b_2}$ in $K_{\mathbf{P}}$. By Theorem 8, there exists an $\{\alpha_{b_1}, \alpha_{b_2}\}$-assignment g defined on $\{x_1, \ldots, x_n\}$ such that $t(g) \not\leq t'(g)$ in $K_{\mathbf{P}}$. Let h be the $\{b_1, b_2\}$-assignment on $\{x_1, \ldots, x_n\}$ naturally induced by g, and let $(c, c') \in C \times C$ be such that $(c, c') \in t(g) \setminus t'(g)$. From property (*), we have that $\phi_t \not\models \phi_{t'}$ is witnessed over $\mathbf{P}_2$ by the assignment $h, (c, c')$.

It remains to show that the translation $t \to \phi_t$ can be computed in polynomial time. Let $s(t)$ denote the size $|\phi_t|$ of a term t. We prove that for a bounded-depth term t, the size $s(t)$ is polynomial in $|t|$, the size of t, which suffices. By inspection of the translation $t \to \phi_t$, there exist natural numbers L, B, E that are polynomial in $|t|$ such that: for a term $t = x_i$, it holds that $s(t) \leq L$; for a term $t = t_1 \wedge \cdots \wedge t_k$ or a term $t = t_1 \vee \cdots \vee t_k$, it holds that $s(t) \leq B(s(t_1) + \cdots + s(t_k)) + E$.

Now, we define the function u recursively as follows: $u(0, n) = Ln$; and, $u(d+1, n) = Bnu(d, n) + E$. We prove the following claim: For all terms t, it holds that $s(t) \leq u(d, n)$, where d is the depth of t and n is the number of leaves (that is, the number of variable occurrences) of t. This suffices, as for each fixed d, the function u can be viewed as a polynomial in L, B, E, and n.

For a term $t = x_i$, we have $d = 0$ and $n = 1$, and that the claim holds is clear from our choice of L. Now, we assume that the claim is true for a depth $d \geq 0$, and we consider a term $t = t_1 \wedge \cdots \wedge t_k$ or a term $t = t_1 \vee \cdots \vee t_k$ having depth $d + 1$. Let n_i denote the number of leaves of t_i, and let d_i denote the depth of t_i; for each i, we have $d_i \leq d$. A direct computation shows that $s(t) \leq u(d+1, n)$. □

From Pentagon Entailment to Containment of pp-Formulas. We now prove Theorem 6.

Proof (Theorem 6). Let $\mathbf{A} = (A, \alpha, \beta, \gamma, \mathcal{D})$ be the finite relational structure defined by the finite algebra $\mathbb{A} \in \mathcal{V}(\mathbb{A}_{\mathbf{B}})$ with universe A, the congruences $\alpha, \beta, \gamma \in \mathrm{Con}(\mathbb{A})$, and the finite set of relations $\mathcal{D}$ from Lemma 2. Notice that $\mathbf{A}$ is such that $\mathsf{PPCON}(\mathbf{A}) \in \mathsf{PPCON}(\mathbb{A})$. We claim that $\mathsf{PPCON}(\mathbf{A})$ is coNP-hard, which implies that $\mathsf{PPCON}(\mathbf{B})$ is coNP-hard by Theorem 3. The coNP-hardness of $\mathsf{PPEQ}(\mathbf{B})$ follows from Proposition 1. Let $\mathcal{P} = \{\mathbf{P}_1, \ldots, \mathbf{P}_{|\mathcal{P}|}\}$ be the finite interesting family of pentagons in Lemma 2.

Let ϕ be a sorted pp-formula, and let $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_m\}$ be the variables of first and second sort in ϕ, respectively. We let $\{x_1, \ldots, x_{n'}\}$ and $\{y_1, \ldots, y_{m'}\}$ be the free variables of first and second sort in ϕ, respectively, where $n' \leq n$ and $m' \leq m$. We construct a pp-formula ϕ' on $\mathbf{A}$, as follows. For each variable z in ϕ, we introduce a fresh variable z'; z' is existentially quantified in ϕ' if and only if z is existentially quantified in ϕ. If ϕ contains the constraint

$x_i = x_j$ for some $1 \leq i, j \leq n$, then ϕ' contains the conjunct $\beta(x'_i, x'_j)$; if ϕ contains the constraint $y_i = y_j$ for some $1 \leq i, j \leq m$, then ϕ' contains the conjunct $\gamma(y'_i, y'_j)$; if ϕ contains the constraint $R(x_i, y_j, y_k)$ for some $1 \leq i \leq n$ and $1 \leq j, k \leq m$, then ϕ' contains the conjunct

$$(\exists w'_1)(\exists w'_2)(\beta(w'_1, x'_i) \wedge \beta(w'_2, x'_i) \wedge \gamma(w'_1, y'_j) \wedge \gamma(w'_2, y'_k) \wedge \alpha(w'_1, w'_2)), \tag{3}$$

where w'_1 and w'_2 are fresh variables; finally, ϕ' contains the conjunct

$$\Delta_{n+m+k}(x'_1, \ldots, x'_n, y'_1, \ldots, y'_m, w'_1, \ldots, w'_k), \tag{4}$$

where Δ_{n+m+k} is the pp-definition on $\mathbf{A}$ of the relation D_{n+m+k} as in Lemma 2 and $\{w'_1, \ldots, w'_k\}$ is the set of fresh variables introduced in conjuncts of type (3). We arrange ϕ' so that the existential quantifiers introduced in conjuncts of type (3) appear at the start of ϕ'.

The desired reduction is the composition of the reduction r given by Theorem 9 and the mapping $(\phi, \psi) \mapsto (\phi', \psi')$. The construction is feasible in polynomial time by Lemma 2, and it is possible to check its correctness by appealing to Lemma 2 and the special character of the pentagons found in $\mathcal{P}$. □

Acknowledgements. Hubie Chen is supported by the Spanish program "Ramon y Cajal" and MICINN grant TIN2010-20967-C04-02. Matt Valeriote acknowledges the support of the Natural Sciences and Engineering Research Council of Canada.

References

1. Atserias, A.: Conjunctive Query Evaluation by Search-Tree Revisited. Theoretical Computer Science 371(3), 155–168 (2007)
2. Barto, L.: CD implies NU (2010) (manuscript)
3. Barto, L., Kozik, M.: Constraint satisfaction problems of bounded width. In: Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, pp. 595–603 (2009)
4. Berman, J., Idziak, P., Marković, P., McKenzie, R., Valeriote, M., Willard, R.: Varieties with Few Subalgebras of Powers. Transactions of the American Mathematical Society 362(3), 1445–1473 (2010)
5. Bodnarchuk, V., Kaluzhnin, L., Kotov, V., Romov, B.: Galois Theory for Post Algebras. I, II. Cybernetics 5, 243–252, 531–539 (1969)
6. Bova, S., Chen, H., Valeriote, M.: On the Expression Complexity of Equivalence and Isomorphism of Primitive Positive Formulas. Theory of Computing Systems, doi:10.1007/s00224-010-9302-7
7. Bulatov, A., Jeavons, P., Krokhin, A.: Classifying the Complexity of Constraints using Finite Algebras. SIAM Journal on Computing 34(3), 720–742 (2005)
8. Bulatov, A.: Tractable Conservative Constraint Satisfaction Problems. In: Proceedings of 18th IEEE Symposium on Logic in Computer Science (LICS 2003), pp. 321–330 (2003)
9. Bulatov, A.: A Dichotomy Theorem for Constraint Satisfaction Problems on a 3-element Set. Journal of the ACM 53 (2006)

10. Bulatov, A., Valeriote, M.: Recent Results on the Algebraic Approach to the CSP. In: Creignou, N., Kolaitis, P.G., Vollmer, H. (eds.) Complexity of Constraints. LNCS, vol. 5250, pp. 68–92. Springer, Heidelberg (2008)
11. Freese, R., McKenzie, R.: Commutator Theory for Congruence Modular Varieties. London Mathematical Society Lecture Note Series, vol. 125. Cambridge University Press, Cambridge (1987)
12. Geiger, D.: Closed Systems of Functions and Predicates. Pacific Journal of Mathematics 27, 95–100 (1968)
13. Hobby, D., McKenzie, R.: The Structure of Finite Algebras. Contemporary Mathematics, vol. 76. American Mathematical Society, Providence (1988); Revised edition: 1996
14. Idziak, P., Markovic, P., McKenzie, R., Valeriote, M., Willard, R.: Tractability and Learnability Arising from Algebras with Few Subpowers. In: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science, LICS (2007)
15. Hunt III, H.B., Rosenkrantz, D.J., Bloniarz, P.A.: On the Computational Complexity of Algebra on Lattices. SIAM Journal on Computing 16(1), 129–148 (1987)
16. Kearnes, K., Kiss, E., Valeriote, M.: Minimal Sets and Varieties. Transactions of the American Mathematical Society 350(1), 1–41 (1998)
17. Kolaitis, P., Vardi, M.: Conjunctive-Query Containment and Constraint Satisfaction. Journal of Computer and System Sciences 61, 302–332 (2000)
18. Papadimitriou, C., Yannakakis, M.: On the Complexity of Database Queries. Journal of Computer and System Sciences 58(3), 407–427 (1999)
19. Vardi, M.: The Complexity of Relational Query Languages. In: Proceedings of the 18th Annual ACM Symposium on Theory of Computing, STOC (1982)

Guarded Negation

Vince Bárány[1,⋆], Balder ten Cate[2,⋆⋆] and Luc Segoufin[3]

[1] University of Warsaw
[2] UC Santa Cruz
[3] INRIA and ENS Cachan, LSV

Abstract. We consider restrictions of first-order logic and of fixpoint logic in which all occurrences of negation are required to be guarded by an atomic predicate. In terms of expressive power, the logics in question, called GNFO and GNFP, extend the guarded fragment of first-order logic and guarded least fixpoint logic, respectively. They also extend the recently introduced unary negation fragments of first-order logic and of least fixpoint logic.

We show that the satisfiability problem for GNFO and for GNFP is 2ExpTime-complete, both on arbitrary structures and on finite structures. We also study the complexity of the associated model checking problems. Finally, we show that GNFO and GNFP are not only computationally well behaved, but also model theoretically: we show that GNFO and GNFP have the tree-like model property and that GNFO has the finite model property, and we characterize the expressive power of GNFO in terms of invariance for an appropriate notion of bisimulation.

1 Introduction

Modal logic is well known for its "robust decidability": not only are basic decision problems such as satisfiability, validity and entailment decidable, but the decidability of these problems is preserved under various natural variations and extensions to the syntax and semantics of modal logic (e.g., addition of fixpoint operators, backward modalities, nominals; restriction to finite structures). As observed by Vardi [14], this robust decidability is intimately linked to the fact that modal logic has a combination of three properties, namely (i) the *tree model property* (if a formula has a model, it has a model which is a tree), (ii) *translatability into monadic second-order logic* (MSO), and thereby into tree automata and, (iii) the *finite model property* (every satisfiable modal formula is satisfied in a finite structure). The decidability of satisfiability (on arbitrary structures and on finite structures) follows immediately from these three properties. However, we should note here that the two way μ-calculus (the extension of modal logic with fixpoint operators and backward modalities) lacks the finite model property, and hence the decidability of satisfiability on finite structures for this logic involves a separate (non trivial) argument [5].

The properties (i), (ii) and (iii) described above can be viewed as a semantic explanation for the robust decidability of modal logic. Given that modal logic can be viewed

⋆ First author was supported by ERC Starting Grant Sosna.
⋆⋆ The second author was supported by NSF grant IIS-0905276.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 356–367, 2011.

as a syntactic fragment of first-order logic, it is also natural to ask for syntactic explanations: *what syntactic features of modal formulas (viewed as first-order formulas) are responsible for their good behavior? And can we generalize modal logic, preserving these features, while at the same time dropping inessential restrictions inherent in modal logic (such as the fact that it can only describe structures with unary and binary relations)?*

Several answers to these questions have been proposed. The first one is to consider the two variable fragment of first-order logic, which is decidable and has the finite model property [12]. Unfortunately, this observation does not go very far towards explaining the robust decidability of modal logic, since it seems impossible to extend the two variable fragment with a fixpoint mechanism while maintaining decidability [9].

The second proposal is to consider logics with guarded quantifications. The *guarded fragment* of first-order logic (GFO) consists of FO formulas in which all quantifiers are "guarded" by atomic predicates. It was introduced in [1]. It has a natural extension with fixpoint operators (GFP) that extends the two-way μ-calculus [10]. Both GFO and GFP have the tree-like model property (if a formula has a model, it has one of bounded tree width), they can be interpreted into MSO (each formula can be transformed into a tree automaton recognizing tree decompositions of its models of bounded tree width) and GFO has the finite model property [1,8]. Finite satisfiability of GFP was only recently proved decidable in [2].

The third, and most recent proposal is based on unary negation. Unary negation first-order logic (UNFO) restricts first-order logic by constraining the use of negation to subformulas having at most one free variable (and viewing universal quantification as a defined connective). Unary negation fixpoint (UNFP) is the natural extension of UNFO using monadic fixpoints. Again, UNFO generalizes modal logic, and UNFP generalizes the two-way μ-calculus. Both UNFO and UNFP have the tree-like model property, they can be interpreted into MSO and UNFO has the finite model property [13]. Decidability of finite satisfiability for UNFP was also established in [13].

The three extensions of modal logics presented above are incomparable in terms of expressive power. In particular there are properties expressible in UNFO that are not expressible in GFO and vice-versa. In this paper we unify the unary negation and guarded quantification approaches by introducing guarded negation logics.

Guarded negation first-order logic (GNFO) restricts FO by requiring that all occurrences of negation are of the form $\alpha \wedge \neg\phi$ where the "guard" α is an atomic formula (possibly an equality statement) containing all the free variables of ϕ. We also disallow universal quantification as a primitive connective (though a limited form of universal quantification can be expressed using existential quantification and guarded negation). For instance, GNFO cannot express $x \neq y$ but it can express $R(x, y, z) \wedge x \neq y$. Guarded negation fixpoint (GNFP) extends GNFO with a guarded fixpoint mechanism. In terms of expressive power, GNFO forms a strict extension of both UNFO and GFO.

We show that our guarded negation logics have the same desirable properties as modal logics, unary negation logics and guarded logics. In particular, the satisfiability problem for GNFO and GNFP is decidable, both on arbitrary structures and on finite structures. These problems are all 2ExpTime-complete, even for a fixed finite schema (recall that satisfiability of GFO is in ExpTime when the schema is fixed). We also study

the (combined) complexity of the model checking problem of GNFO and GNFP. The problem is $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$-complete for GNFO. In the case of GNFP, it is hard for P^{NP} and contained in $\mathrm{NP}^{\mathrm{NP}} \cap \mathrm{coNP}^{\mathrm{NP}}$. Note that a similar gap between the upper bound and the lower bound exists for GFP and the μ-calculus, where the complexity of model checking is known to lie between PTime and NP $\cap$ coNP [4]. Recall that the model checking problem of GFO is PTime-complete [4]. Our proofs are based on reductions to the model checking problem for UNFO and UNFP. Finally, we show that GNFO and GNFP have the tree-like model property, and that GNFO has the finite model property, and we characterize the expressive power of GNFO in terms of invariance for an appropriate notion of bisimulation.

The most difficult result is the decidability of satisfiability on finite structures. For GNFO, we give a reduction to testing whether a union of conjunctive queries is implied by a guarded formula, recently shown decidable in [3]. In the case of GNFP, we make a reduction to the decidability of finite satisfiability of GFP, recently proved in [2].

Related work. GNFO and GNFP form decidable extensions of GFO and GFP. Other decidable extensions of GNFO and GNFP have been considered in the past, most notably the *clique-guarded* fragment (and the related *packed* fragment, as well as the weaker *loosely-guarded* fragment) of first-order logic, and of least fixpoint logic [6]. The logics GNFO and GNFP we propose here are incomparable in expressive power to the clique guarded fragments. We leave open the question whether a decidable common generalization exists.

2 Preliminaries

Structures and formulas. We are working on relational structures. We assume given a relational schema τ consisting of a finite set of relation symbols, each having an associated arity. By the *arity of a schema*, we mean the maximal arity of its relations. A *structure* (or *model*) M over a relational schema τ consists of a set $dom(M)$, the *domain* of M, together with an interpretation of each relation symbol $R \in \tau$ as a k-ary relation over $dom(M)$ for k the arity of R according to τ. A structure M is said to be *finite* if $dom(M)$ is finite. If a tuple of elements $\bar{a}$ from $dom(M)$ belongs to the interpretation of a relation symbol R, then we say that $R(\bar{a})$ is a *fact* of M. A tuple (or set) of elements of M is *guarded* if it is a singleton or all its components (elements) occur among those in a fact of M.

We assume familiarity with first-order logic, FO, and least fixpoint logic, LFP, over relational structures. We use classical syntax and semantics for FO and LFP. We write $\phi(\bar{x})$ to denote the fact that the free variables of ϕ are exactly the variables in $\bar{x}$. We also write $M \models \phi(\bar{u})$ or $M, \bar{u} \models \phi(\bar{x})$ for the fact that the tuple $\bar{u}$ of elements of the model M makes the formula $\phi(\bar{x})$ true in M. The *size of a formula* ϕ, denoted by $|\phi|$, is the number of symbols needed to write down the formula.

Conjunctive queries. A conjunctive query (CQ) is a first-order formula of the form $\exists \bar{x} \alpha$ where α is a conjunction of positive atomic formulas (including equalities). A union of conjunctive queries (UCQ) is a disjunction of CQs. A positive-existential query is an FO formula built using disjunction, conjunction and existential quantification only.

Every positive-existential query can be transformed in a UCQ at the cost of a possible exponential blow-up. Positive-existential queries belong to GNFO, even to UNFO. The width of a CQ is the number of variables occurring in it, and the width of a UCQ is the maximum width of its CQs. The height of a UCQ is the maximum size of its CQs.

GNFO. We define GNFO, *guarded negation* FO, as the fragment of FO given by the following grammar, where R ranges over predicate symbols, and $\alpha(\bar{x}\bar{y})$ is an atomic formula (possibly an equality statement).

$$\varphi ::= \quad R(\bar{x}) \mid x = y \mid \exists x\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \alpha(\bar{x}\bar{y}) \wedge \neg\varphi(\bar{y}) \tag{1}$$

Hence the logic can only negate a subformula if all its free variables are "guarded" by some fact, or if the subformula has at most one free variable (in which case one can use an equality statement of the form $x = x$ or $y = y$ as the guard). For example, $x \neq y$ is not a formula of GNFO but $R(x, y, z) \wedge x \neq y$ is.

We say that a formula of GNFO is in *GN-normal form* if, in its syntax tree, no disjunction is directly below an existential quantifier or a conjunction, and no existential quantifier is directly below a conjunction sign. Every GNFO formula can be brought into GN-normal form, at the cost of an exponential increase in length and linear increase in the number of variables, using the following equivalences as rewrite rules (where x' is a variable not occurring in ψ):

$$\exists x(\phi \vee \psi) \simeq \exists x\phi \vee \exists x\psi,\ \phi \wedge (\psi \vee \chi) \simeq (\phi \wedge \psi) \vee (\phi \wedge \chi),\ (\exists x\phi) \wedge \psi \simeq \exists x'(\phi[x'/x] \wedge \psi)$$

The appeal of the GN-normal form is that it highlights the fact that GNFO formulas can be naturally viewed as being built up from atomic formulas using guarded negation, and unions of conjunctive queries. Indeed, the GNFO formulas in GN-normal form are precisely generated by the following recursive definition:

$$\varphi ::= \quad R(\bar{x}) \mid x = y \mid \alpha(\bar{x}\bar{y}) \wedge \neg\varphi(\bar{y}) \mid q[\varphi_1/U_1, \ldots, \varphi_s/U_s] \tag{2}$$

where q is a UCQ using relation symbols $U_1, \ldots, U_s$, and $\varphi_1, \ldots, \varphi_s$ are formulas (generated by the same recursive definition) with the appropriate number of free variables corresponding to the relation symbols they replace. Here, $q[\varphi_1/U_1, \ldots, \varphi_s/U_s]$ is the result of replacing in q all subformulas of the form $U_i(\bar{x})$ with $i \leq s$ by $\phi_i(\bar{x})$.

A formula of GNFO is said to be *of width* k if, when brought into GN-normal form in the way described above, it uses at most k variables (or equivalently, is built up using UCQs q of width at most k). We denote by GNFO^k all GNFO formulas of width k.

GNFO extends GFO and UNFO. GNFO generalizes the logic UNFO, studied in [13], which only allows the negation of formulas having at most one free variable. It also generalizes the *guarded fragment of first-order logic* (GFO). The logic GFO is the fragment of FO defined by the following grammar, where, again, $\alpha(\bar{x}\bar{y}\bar{z})$ is an atomic formula (possibly an equality statement):

$$\varphi ::= R(\bar{x}) \mid x = y \mid \varphi \vee \varphi | \varphi \wedge \varphi | \neg\varphi \mid \exists\bar{x}\ \alpha(\bar{x}\bar{y}\bar{z}) \wedge \varphi(\bar{x}\bar{y}) \mid \forall\bar{x}\ \alpha(\bar{x}\bar{y}\bar{z}) \rightarrow \varphi(\bar{x}\bar{y})$$

It is straightforward to check that:

Proposition 1. *Every* GFO *sentence is equivalent to a* GNFO *sentence, via a polynomial time transformation.*[1]

[1] This is only true for *sentences*, as $\neg R(xy)$ is in GFO but not expressible in GNFO.

Proof. Let φ be any GFO sentence. We may assume that φ does not contain universal quantifiers, by using negation and (guarded) existential quantification instead. Since φ is a sentence, every subformula $\vartheta(\bar{x})$ is in the scope of an (inner-most) guarded existential quantifier $\exists \bar{x}\bar{u}(\alpha_\vartheta(\bar{x}\bar{u}) \wedge \cdots)$. We replace in φ each negated subformula $\vartheta(\bar{x}) = \neg\psi(\bar{x})$ by $\alpha_\vartheta(\bar{x}\bar{u}) \wedge \neg\psi(\bar{x})$ to obtain the desired equivalent GNFO-sentence. □

Example 1. The GNFO sentence $\delta = \exists xy\big(E(x,y) \wedge \neg\exists uvw(E(x,u) \wedge E(u,v) \wedge E(v,w) \wedge E(w,y))\big)$ is not equivalent to any GFO sentence or to any UNFO sentence, even on undirected graphs. This is because δ defines a property that is not invariant under guarded bisimulation (which, incidentally, amounts to ordinary bisimulation in case of undirected graphs), as can be easily verified, nor is it invariant under "UN-bisimulation" as befits UNFO formulas, cf. [13].

3 The Satisfiability Problem for GNFO

We show in this section how to reduce the (finite) satisfiability problem for GNFO to the problem of testing whether a GFO formula entails (on finite structures) a UCQ. The latter problem is also known as the problem of query answering against a GFO theory, and it has been solved in [3]. To streamline the presentation, we will allow the possibility of zero-ary relation symbols.

Lemma 1. *To every $\varphi(\bar{x}) \in \mathsf{GNFO}[\tau]$ one can associate in polynomial time a companion formula $\psi(\bar{x}) \in \mathsf{GNFO}[\tau \uplus \sigma]$ of the form*

$$\psi(\bar{x}) = \underbrace{S(\bar{x}) \wedge \bigwedge_j \forall\, \bar{z}\bar{u}.R_j(\bar{z}\bar{u}) \to q_j(\bar{z})}_{\psi^+} \quad \wedge \quad \underbrace{\bigwedge_i \forall\, \bar{z}\bar{u}.T_i(\bar{z}\bar{u}) \to \neg p_i(\bar{z})}_{\psi^-} \tag{3}$$

where σ comprises the new relation symbols occurring as S, R_j or T_i, where the q_j's and p_i's are positive-existential, $\mathrm{width}(\psi) = \mathrm{width}(\varphi)$ *and such that* $\varphi \leftrightarrow \exists S\, \exists \overline{T}\, \psi$.

Proof. Given a GNFO-formula φ consider an inner-most occurrence of a guarded negation $R(\bar{z}\bar{u}) \wedge \neg q(\bar{z})$ as a subformula of φ. Then $q(\bar{z})$ is necessarily positive existential. Let T be a new predicate symbol of the same arity as R. We substitute $T(\bar{z}\bar{u})$ in the input formula for the subformula $R(\bar{z}\bar{u}) \wedge \neg q(\bar{z})$, and add the following as conjuncts to ψ^+ and ψ^-, according to their kind.

$$\begin{aligned} &\forall\, \bar{z}\bar{u}.T(\bar{z}\bar{u}) \to \neg q(\bar{z}) \\ &\forall\, \bar{z}\bar{u}.T(\bar{z}\bar{u}) \to R(\bar{z}\bar{u}) \\ &\forall\, \bar{z}\bar{u}.R(\bar{z}\bar{u}) \to T(\bar{z}\bar{u}) \vee q(\bar{z}) \end{aligned}$$

Inner-most *equality-guarded* negations $z = u \wedge \neg q(z,u)$ are handled in a similar fashion. Again, $q(z,u)$ must be positive-existential. We choose a new unary relation symbol T, replace the subformula in question by $z = u \wedge T(z)$, and add $\forall\, z.T(z) \to \neg q[u/z]$ and $\forall\, z.T(z) \vee q[u/z]$ as conjuncts to the normal form.

Proceeding in this manner from the inside-out we eliminate all guarded negations until the original input formula is reduced to a single positive-existential formula $p(\bar{x})$ (in the extended signature). Finally we replace $p(\bar{x})$ with $S(\bar{x})$ where S is an appropriate

new predicate symbol and add $\forall\, \bar{x}.S(\bar{x}) \rightarrow p(\bar{x})$ as conjunct to the normal form, which is thus finalized. It is now easy to verify the correctness of this transformation. □

We may assume wlog. that the positive-existential formulas q_j of (3) are in prenex normal form, i.e. $q_j(\bar{z}) = \exists \bar{u} \xi_j(\bar{z}, \bar{v})$. Also note that each conjunct $\forall\, \bar{z}\bar{u}.T_i(\bar{z}\bar{u}) \rightarrow \neg p_i(\bar{z})$ of (3) is the negation of a positive-existential sentence $\exists\, \bar{z}\bar{u}.T_i(\bar{z}\bar{u}) \wedge p_i(\bar{z})$. Therefore, the entire ψ^- of (3) can be conceived as the negation of a single positive-existential sentence q. This leads us to the following equivalent formula.

$$\underbrace{S(\bar{x}) \wedge \bigwedge_j \left(\forall\, \bar{z}\bar{u}.R_j(\bar{z}\bar{u}) \rightarrow \exists \bar{v} \xi_j(\bar{z}\bar{v}) \right)}_{\psi^+} \wedge \neg \underbrace{\bigvee_i \left(\exists\, \bar{z}\bar{u}.T_i(\bar{z}\bar{u}) \wedge p_i(\bar{z}) \right)}_{q} \tag{4}$$

Observe next that without affecting satisfiability of (4) we may introduce new atoms guarding the existential quantifiers in ψ^+ thus obtaining a GFO-formula

$$\psi^* = S(\bar{x}) \wedge \bigwedge_j \left(\forall\, \bar{z}\bar{u}.R_j(\bar{z}\bar{u}) \rightarrow \exists \bar{v} Q_j(\bar{z}\bar{v}) \wedge \xi_j(\bar{z}\bar{v}) \right)$$

where the Q_j's are distinct new relation symbols of appropriate arity. Then, $\psi^* \models \psi^+$ and, conversely, every model of ψ^+ has an expansion modeling ψ^*.

The entire transformation of an input GNFO-formula φ to the equi-satisfiable $\psi^* \wedge \neg q$, with ψ^* in GFO and q positive existential, can be performed in polynomial time and only results in a polynomial blowup in the signature of the latter normal form. In a final transformation step, which may require at most exponential time, the positive-existential sentence q can be converted to an equivalent Boolean UCQ q^*. In general q^* may be comprised of exponentially many CQs each of size at most $|q|$. Summing up all the reduction steps we obtain:

Proposition 2. *For each* $\varphi(\bar{x}) \in \mathsf{GNFO}[\tau]$ *one can compute in exponential time a* GFO*-formula* $\psi^*(\bar{x})$ *and* UCQ $q^* = \bigvee_l Q_l$*, both of signature* $\tau \uplus \{\bar{T}\}$*, such that*

$$\varphi \longleftrightarrow \exists \bar{T} \left(\psi^* \wedge \neg q^* \right) \tag{5}$$

is valid, and that $|\psi^*| = \mathcal{O}(|\varphi|)$ *and* $\mathrm{height}(q^*) = \max_l |Q_l| \leq |\varphi|$.

We now summarize the main results of [3]. Later we will build on key elements of the construction of [3], stated below as Lemmas 2 and 3 and Theorem 4, from which the following Theorem 1 can be directly derived.

Theorem 1 ([3]). *Given a* GFO*-formula* ψ *and a* UCQ q *of height* h *it is decidable in time* $|q| \cdot 2^{(h|\psi|)^{\mathcal{O}(h|\psi|)}}$ *whether or not* $\psi \wedge \neg q$ *is satisfiable; and if* $\psi \wedge \neg q$ *has a model then it has a finite model of size* $2^{(h|\psi|)^{\mathcal{O}(h|\psi|)}}$

By combining Theorem 1 with the estimates of Proposition 2 we derive the complexity of satisfiability for GNFO, as well as its finite model property.

Theorem 2. *1. The satisfiability problem for* GNFO *is* 2ExpTime*-complete.*

2. Every satisfiable GNFO*-sentence* φ *has a finite model of size* $2^{2^{|\varphi|^{\mathcal{O}(1)}}}$.

The 2ExpTime lower bound follows immediately from the fact that satisfiability for UNFO is already hard for 2ExpTime [13,7]. It holds even if the schema is fixed (recall that when the schema is fixed the complexity of satisfiability for GFO is ExpTime-complete).

4 The Satisfiability Problem for GNFP

In a nutshell, GNFP is the extension of GNFO with guarded fixpoints. We show here that both satisfiability and finite satisfiability are decidable for GNFP.

GNFP. In order to define GNFP we introduce extra predicate variables, which will serve for computing fixpoints. We denote the predicates given by the relational schema by P, Q, R, S etc. and the predicate variables serving for computing the fixpoints by X, Y, Z etc. However the fixpoint predicates are not permitted to be used as guards. For instance $R(\bar{x}) \wedge \neg Y(\bar{x})$ is allowed but $Y(\bar{x}) \wedge \neg R(\bar{x})$ is not. Formulas of GNFP$[\tau]$, we omit the schema τ when it is clear from the context, pertain to the following syntax where R is any relational symbol in τ, $\alpha(\bar{x}\bar{y})$ is an atomic formula (possibly an equality statement), and $\sigma \subseteq \tau$.

$$\phi ::= R(\bar{x}) \mid \texttt{x=y} \mid X(\bar{x}) \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \exists x\, \phi \mid \alpha(\bar{x}\bar{y}) \wedge \neg\phi(\bar{x}) \mid \mu_{Z,\bar{z}}[\texttt{guarded}_\sigma(\bar{z}) \wedge \phi(\bar{Y}, Z, \bar{z})](\bar{x}) \quad \mid \quad \nu_{Z,\bar{z}}[\texttt{guarded}_\sigma(\bar{z}) \wedge \phi(\bar{Y}, Z, \bar{z})](\bar{x})$$

Here μ and ν stand for least- and greatest fixpoints, respectively, and it is further required that in the matrix $\phi(\bar{Y}, Z, \bar{z})$ of a fixpoint formula i) the fixpoint variable Z occurs only positively (i.e. always under an even number of negations) and never as a guard; ii) no first-order parameters (i.e., free variables other than those $\bar{z}$ bound by the fixpoint operator) are permitted; and iii) free fixpoint variables $\bar{Y}$ are allowed, to enable nesting of fixpoint declarations. The clause $\texttt{guarded}_\sigma(\bar{z})$ signifies the requirement that all tuples belonging to a predicate defined by a fixpoint construct must be guarded by a relational atom of the underlying structure. The clause $\texttt{guarded}_\sigma(\bar{z})$ can be understood in either of two ways: as a syntactic element (keyword) signifying this intended semantics, or as a formula defining guardedness by a disjunction of existentially quantified relational atoms (allowing relation symbols from σ as well as equality) involving all of the variables $\bar{z}$. Note that while the definitions of least- and greatest fixpoint formulas are symmetric, the two operators are *not* each others duals. This is due to the fact that the dualization of ν may introduce negations that are not guarded.

Since GNFP can be seen as a syntactic fragment of least fixed point logic LFP, we omit the definition of the semantics, cf. [11].

The definition of GN-normal form that we gave for GNFO formulas applies to GNFP as well. Formulas of GNFP in GN-normal form can be naturally thought of as being built up from atomic formulas using (i) guarded negation, (ii) unions of conjunctive queries, and (iii) fixpoint operators. As in the case of GNFO, the *width* of a GNFP-formula is the number of variables it contains after being put in GN-normal form and we let GNFP^k denote the set of GNFP-formulas of width k.

GNFP extends GFP and UNFP. Syntactically, GNFP generalizes the logic UNFP studied in [13], which only allows the negation of formulas having at most one free

variable, and only unary fixpoints. GNFP also generalizes the *guarded fragment of fixpoint logic* (GFP) [10]. The logic GFP is the fragment of LFP defined as for GNFP but replacing the first-order part of the syntax based on GNFO by the syntax of GFO. It is not immediate from the syntax, but it is easy to check by induction building on Proposition 1 that GNFP generalizes GFP.

Proposition 3. *Every sentence of* GFP *is equivalent to a sentence of* GNFP*, via a polynomial time transformation.*

The aim of this section is to establish the following main result.

Theorem 3. *It is decidable whether a sentence of* GNFP *has a model and whether it has a finite model. Both of these problems are 2*ExpTime*-complete.*

The proof of Theorem 3 is a reduction to the (finite) satisfiability of GFP: given a formula of GNFP we construct a formula of GFP whose (finite) satisfiability is equivalent to the one of the initial formula and we then apply known results on (finite) satisfiability for GFP [10,2]. Before we describe the reduction, we start with some useful notation and some preliminary results taken from [3].

Acyclicity and treeification. We say that a conjunctive query is *acyclic* if it is semantically equivalent to a formula of GFO built with only conjunction and existential quantification. For instance the query $\exists yzw\, T(x,y,z) \wedge T(x,w,z) \wedge E(x,y)$ is acyclic because it is equivalent to the guarded formula $\exists yzT(x,y,z) \wedge E(x,y) \wedge (\exists wT(x,w,z))$. It is easy to check that this definition is equivalent to acyclicity (in the hypergraph sense) of the hypergraph induced by the atoms of the query with its variables as vertices.

Definition 1. *Given a schema τ, the* τ-treeification $\Lambda_q^\tau(\bar{x})$ *of a positive existential query $q(\bar{x})$ over τ is the* UCQ *consisting of the disjunction of all those* acyclic *CQs over τ (modulo renaming of bound variables) that imply q and that are minimal (in the sense that removing any atomic formula would render it non-acyclic or not implying q).*

Consider for instance $q(x) = \exists yzw\, E(x,y) \wedge E(y,z) \wedge E(z,w) \wedge E(w,x)$. Then $\Lambda_q^{\{E\}} = E(x,x) \vee \exists y\, E(x,y) \wedge E(y,x)$. Indeed, the only acyclic queries implying $q(x)$ are obtained by identifying some of its variables resulting in either a reflexive edge on x or a pair of inverse edges. If the schema is $\{E,T\}$ where T is a ternary predicate the treeification of $q(x)$ has a number of additional disjuncts corresponding to various triangulations of $q(x)$, such as $\exists yzw\, T(x,y,z) \wedge T(x,w,z) \wedge E(x,y) \wedge E(y,z) \wedge E(z,w) \wedge E(w,x)$. It can be shown that each disjunct in the treeification of any CQ in whatever schema contains at most three times as many atoms as the CQ itself [3] leading to the following observations.[2]

Lemma 2. *Consider a schema τ having r many predicate symbols of maximal arity w. Let $q(\bar{x})$ be a UCQ of height h over τ. Then $\Lambda_q^\tau(\bar{x})$ has width w, length $r^{\mathcal{O}(h)}(hw)^{\mathcal{O}(hw)}$, height $\mathcal{O}(h)$, and can be constructed in time $|q|r^{\mathcal{O}(h)}(hw)^{\mathcal{O}(hw)}$.*

[2] These figures constitute a slight refinement of those offered in [3, Lemma 10].

Guarded bisimulation and covers. Guarded bisimulations [6] are, roughly speaking, strategies for Duplicator in Ehrenfeucht-Fraïssé games played on structures M and N, with positions restricted to $(\bar{a}, \bar{b})$ such that $\bar{a}$ is guarded in M and $\bar{b}$ guarded in N (cf. also Definition 3 below). Guarded bisimilarity implies GFP-indistinguishability [6]. There is an associated notion of guarded unraveling, which, given a structure M, provides an acyclic structure M^* that is guarded bisimilar to M, thus exhibiting the tree-like model property of GFP [6]. Note that M^* is in general infinite and, because it is acyclic, over M^* every conjunctive query is equivalent to its treeification (indeed, this is one justification for the name "treeification"). An analogue of guarded unraveling for finite structures was provided in [3] in the form of a construction, for all n, of a finite companion $M^{(n)}$ of any finite structure M such that $M^{(n)}$ is guarded bisimilar to M and $M^{(n)} \models q \leftrightarrow \Lambda_q^\tau$ for all UCQs q of width at most n.

Definition 2. A guarded bisimilar cover $\pi\colon N \xrightarrow{\sim} M$ *is an onto homomorphism* $\pi\colon N \to M$ *inducing a guarded bisimulation* $\{(\bar{b}, \pi(\bar{b})) \mid \bar{b}$ *guarded in* $N\}$. *The cover is* weakly n-acyclic *if for every homomorphism* $h : Q \to N$ *from a structure* Q *on at most* n *elements into* N *there exists an acyclic substructure* M_0 *of* M *(not necessarily induced) with* $\pi(h(Q)) \subseteq M_0$.

The following lemma shows how covers relate to treeification.

Lemma 3. *Let* $\pi : N \xrightarrow{\sim} M$ *be a weakly* n*-acyclic guarded bisimilar cover of* τ*-structures, and let* $q(\bar{x})$ *be a* UCQ *of width at most* n. *Then for every* guarded tuple $\bar{b}$ *(of not necessarily distinct elements) in* N *we have* $N \models q(\bar{b}) \leftrightarrow \Lambda_q^\tau(\bar{b})$.

Theorem 4 (Rosati cover [3]). *For all* $n \in \mathbb{N}$ *every relational structure* M *of schema* τ *admits a weakly* n*-acyclic guarded bisimilar cover* $\pi\colon M^{(n)} \xrightarrow{\sim} M$. *If* M *is finite then* $|M^{(n)}| = |M|^{w^{\mathcal{O}(n)}}$, *where* w *is the arity of* τ, *and* $M^{(n)}$ *can be effectively constructed. We call* $M^{(n)}$ *the* n-th Rosati cover *of* M.

Say that a relation $Z \subseteq M^r$ is *guarded* in M if every tuple $\bar{a} \in Z$ is guarded in M. The following is an immediate consequence of the definitions.

Fact 1. *Consider a weakly* n*-acyclic cover* $\pi : N \xrightarrow{\sim} M$ *and guarded predicates* $Z_1, \ldots, Z_k$ *over* M. *Then* $\pi : (N, W_1, \ldots, W_k) \xrightarrow{\sim} (M, Z_1, \ldots, Z_k)$ *is a weakly* n*-acyclic cover, where* $W_i = \pi^{-1}(Z_i) = \bigcup\{\pi^{-1}(\bar{a}) \mid \bar{a} \in Z_i\}$ *for each* $1 \le i \le k$.

Reduction to (finite) satisfiability for GFP. Let φ be any given GNFP sentence. As a first step, we compute its GN-normal form $\tilde{\varphi}$. Note that $\tilde{\varphi}$ has the following dimensions: $|\tilde{\varphi}| = 2^{\mathcal{O}(|\varphi|)}$, $\mathrm{width}(\tilde{\varphi}) = \mathcal{O}(|\varphi|)$, and $\tilde{\varphi}$ is built up using only UCQs of height at most $|\varphi|$ (as well as guarded negations and fixpoint operators) as in (2).

Next, essentially, our reduction transforms all UCQs occurring in $\tilde{\varphi}$ to their treeification. For every $k \ge 1$, and for every relational schema τ consisting of at most k-ary relations, we define a translation η from $\mathrm{GNFP}^k[\tau]$ sentences in GN-normal form to $\mathrm{GFP}^k[\tau \uplus \{C_k\}]$ sentences, where C_k is a new symbol of arity k, by structural recursion, using the following rules.

$$\eta(\, R(\bar{x})\,) = R(\bar{x}) \qquad (a)$$
$$\eta(\, Z(\bar{x})\,) = Z(\bar{x}) \qquad (b)$$
$$\eta(\, \alpha(\bar{x}\bar{y}) \wedge \neg\psi(\bar{x})\,) = \alpha(\bar{x}\bar{y}) \wedge \neg\,\eta(\psi(\bar{x})) \qquad (c)$$
$$\eta(\, \mu_{Z,\bar{z}}[\,\mathtt{guarded}_\tau(\bar{z}) \wedge \psi(\bar{Y}, Z, \bar{z})\,]\,) = \mu_{Z,\bar{z}}[\,\mathtt{guarded}_\tau(\bar{z}) \wedge \eta(\psi(\bar{Y}, Z, \bar{z}))\,] \qquad (d)$$
$$\eta(\, \nu_{Z,\bar{z}}[\,\mathtt{guarded}_\tau(\bar{z}) \wedge \psi(\bar{Y}, Z, \bar{z})\,]\,) = \nu_{Z,\bar{z}}[\,\mathtt{guarded}_\tau(\bar{z}) \wedge \eta(\psi(\bar{Y}, Z, \bar{z}))\,] \qquad (d')$$
$$\eta(\, q[\phi_1/U_1, \ldots, \phi_s/U_s]\,) = \Lambda_q^{\tau \uplus \{U_1,\ldots,U_s,C_k\}}[\eta(\phi_1)/U_1, \ldots, \eta(\phi_s)/U_s] \qquad (e)$$

where in (e) q is a UCQ of signature $\{U_1, \ldots, U_s\}$ disjoint from $\tau \uplus \{\bar{Y}, C_k\}$ and $\phi_1, \ldots, \phi_s \in \text{GNFP}^k[\tau \uplus \{\bar{Y}\}]$, where $\bar{Y}$ enumerates the free fixpoint variables occurring in any of the ϕ_i's, each of which is of a form (a)–(d), and such that $q[\phi_1/U_1, \ldots, \phi_s/U_s]$ is a subformula of $\tilde{\varphi}$. In particular the ϕ_i define guarded relations. It can be readily seen that all formulas in GN-normal form can be decomposed as in (a)–(e) and we have the following bounds and on the translation η.

Lemma 4. *For all* GNFP^k*-formula* φ *of GN-normal form* $\tilde{\varphi}$, $|\eta(\tilde{\varphi})| = 2^{(k|\varphi|)^{\mathcal{O}(1)}}$ *and* $\eta(\tilde{\varphi})$ *can be computed within this time bound, and its width remains* k.

The following key lemma generalizes Lemma 3 and attests to the correctness of our reduction. It is proved by structural induction on formulas, while relying on Fact 1 and Lemma 3 to deal with the cases (d) and (e), respectively.

Lemma 5. *Let* $\pi : N \xrightarrow{\sim} M$ *be a weakly* k*-acyclic guarded bisimilar cover of* $\tau \uplus \{C_k\}$*-structures and* $\phi(\bar{Y}, \bar{x}) \in \text{GNFP}^k[\tau]$ *a formula in GN-normal form with free fixpoint variables* $\bar{Y}$*. Then for every assignment of guarded relations* $\bar{W}$ *on* N *to* $\bar{Y}$ *such that* $W_i = \pi^{-1}(\pi(W_i))$ *for all* $1 \leq i \leq |\bar{Y}|$ *and for every guarded tuple* $\bar{b}$ *in* N *we have:*

$$(N, \bar{W}) \models \eta(\phi)(\bar{b}) \leftrightarrow \phi(\bar{b})\ .$$

Theorem 5. *A* GNFP^k*-sentence* $\tilde{\varphi}$ *in GN-normal form is satisfiable (in the finite) if, and only if,* $\eta(\tilde{\varphi}) \in \text{GFP}^k$ *is satisfiable (in the finite).*

Proof. It is easy to see that for every model M of $\tilde{\varphi}$ its expansion (M, C_k) is a model of $\eta(\tilde{\varphi})$, where each C_k is the complete k-ary relation on M.

Conversely, consider some M a model of $\eta(\tilde{\varphi})$ and its k-th Rosati cover $M^{(k)}$, equally a model of $\eta(\tilde{\varphi})$. Lemma 5 proves that $M^{(k)}$ is, in fact, a model of $\tilde{\varphi}$, and we know from Theorem 4 that if M is finite then so is $M^{(k)}$. □

Both satisfiability [10] and finite satisfiability [2] of GFP sentences have been shown decidable in time $2^{\mathcal{O}(nw^w)}$, where n is the length of the input formula and w is its width. Starting with a GNFP^k sentence φ whose GN-normal form is $\tilde{\varphi}$, we get from Lemma 4 that $|\eta(\tilde{\varphi})| = 2^{(k|\varphi|)^{\mathcal{O}(1)}}$ and that $\eta(\tilde{\varphi})$ is computable within that same time bound, but its width remains k. Theorem 3 now follows from these bounds via Theorem 5.

5 Additional Results

Model checking In this section we study the combined complexity of the model checking problem, where the input consists of a sentence and a structure and the goal is to

decide whether the sentence is true on the structure. It was shown in [13] that the model checking problem for UNFO is $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$-complete, and that the model checking problem for UNFP is in $\mathrm{NP}^{\mathrm{NP}} \cap \mathrm{coNP}^{\mathrm{NP}}$ and P^{NP}-hard. We show that the upper-bounds also apply to GNFO and GNFP. The proof is a reduction to formulas with unary negations by constructing an incidence structure.

Theorem 6. *The model checking problem for* GNFO *is* $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$*-complete. For* GNFP *it is in* $\mathrm{NP}^{\mathrm{NP}} \cap \mathrm{coNP}^{\mathrm{NP}}$ *and hard for* P^{NP}.

Expressive power. We develop an appropriate notion of bisimulation for GNFO and GNFP, and use it to characterize the expressive power of GNFO.

Recall that a tuple of elements of a structure M is said to be *guarded* if there is a fact of M in which all elements from the tuple occur. We denote by $guarded(M)$ the set of all guarded tuples of M. If M and N are structures and $\bar{a}$ and $\bar{b}$ are tuples of elements from $dom(M)$ and $dom(N)$, respectively, then we say that $M, \bar{a}$ and $N, \bar{b}$ are *locally isomorphic* if there is a partial isomorphism $f : M \to N$ such that $f(\bar{a}) = \bar{b}$.

Definition 3. *Let M, N be two structures. A GN-bisimulation (of width $k \geq 1$) is a binary relation $Z \subseteq guarded(M) \times guarded(N)$ such that the following hold for every pair $(\bar{a}, \bar{b}) \in Z$, where $\bar{a} = a_1, \ldots, a_m$ and $\bar{b} = b_1, \ldots, b_n$*

- *$M, \bar{a}$ and $N, \bar{b}$ are locally isomorphic (and in particular, $m = n$)*
- *For every finite set $X \subseteq dom(M)$ (with $|X| \leq k$) there is a partial homomorphism $h : M \to N$ whose domain is X, such that $h(a_i) = b_i$ for all a_i in X, and such that every $\bar{a}' \in guarded(M)$ consisting of elements in the domain of h, the pair $(\bar{a}', h(\bar{a}'))$ belongs to Z.*
- *Likewise in the other direction, where $X \subseteq dom(N)$.*

Note that if X above is restricted to guarded sets then we obtain a definition of guarded bisimulation. We write $M \approx_{GN}^{(k)} N$ if there is a non-empty GN-bisimulation (of width k) between M and N.

Proposition 4. *For $k \geq 1$, if $M \approx_{GN}^{k} N$ then M and N satisfy the same* GNFP^k *sentences. In particular, if $M \approx_{GN} N$ then M and N satisfy the same* GNFP *sentences.*

In fact, GN-bisimulation invariance can be used to *characterize* GNFO.

Theorem 7. GNFO *is the $\approx_{GN}$-invariant fragment of* FO*, and for all $k \geq 1$,* GNFO^k *is the $\approx_{GN}^{k}$-invariant fragment of* FO *(on arbitrary structures).*

Finally, building on Theorem 7 and Theorem 5, we can show the following.

Theorem 8. GNFP *has the tree-like model property.*

6 Discussion

We have provided a logical framework generalizing both GFO and UNFO while preserving their nice properties, in particular decidability of satisfiability. Our results on satisfiability carry over to the validity and entailment problems for GNFO, and likewise

for GNFP, as these problems are all reducible to each other. For instance, a GNFO entailment $\phi(\bar{x}\bar{y}) \models \psi(\bar{x}\bar{z})$ holds if, and only if, for a fresh relation R of appropriate arity $\exists\bar{x}\bar{y}\bar{z}(\phi(\bar{x}\bar{y}) \wedge R(\bar{x}\bar{y}\bar{z}) \wedge \neg\psi(\bar{x}\bar{z})))$ is not satisfiable.

Another immediate consequence of our results is that query answering for unions of conjunctive queries with respect to guarded negation fixpoint theories (i.e., the analogue of Theorem 1 replacing GFO by GNFP) is decidable and 2ExpTime-complete. Furthermore, although our definition of GNFO does not include constant symbols, they could be added without affecting the complexity of these problems, relying on the same technique used in [7].

It would be tempting to further generalize by including the two variable fragment of FO (FO^2). Unfortunately this would lead to undecidability. Actually a simple combination of FO^2 with UNFO already yields undecidability as FO^2 can express the fact that a relation correspond to inequality ($\forall x, y \;\; R(x, y) \leftrightarrow x \neq y$) and the extension of UNFO with inequality is undecidable [13]. Similarly, unconstrained universal quantification leads to undecidability, since every subformula of the form $\neg\psi(\bar{x})$ can be trivially guarded using a fresh relation $R(\bar{x})$, adding $\forall\bar{x}\; R\bar{x}$ as a conjunct to the main formula.

References

1. Andréka, H., van Benthem, J., Németi, I.: Modal languages and bounded fragments of predicate logic. Journal of Philosophical Logic 27, 217–274 (1998)
2. Bárány, V., Bojańczyk, M.: Finite satisfiability for guarded fixpoint logic. Draft available at arXiv:1104.2262v1 [cs.LO] (2011)
3. Bárány, V., Gottlob, G., Otto, M.: Querying the guarded fragment. In: Symposium on Logic In Computer Science, LICS (2010)
4. Berwanger, D., Grädel, E.: Games and model checking for guarded logics. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS (LNAI), vol. 2250, p. 70. Springer, Heidelberg (2001)
5. Bojańczyk, M.: Two-way alternating automata and finite models. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, p. 833. Springer, Heidelberg (2002)
6. Grädel, E.: Decision procedures for guarded logics. In: Ganzinger, H. (ed.) CADE 1999. LNCS (LNAI), vol. 1632, pp. 31–51. Springer, Heidelberg (1999)
7. Grädel, E.: On the restraining power of guards. J. Symb. Logic 64(4), 1719–1742 (1999)
8. Grädel, E.: Why are modal logics so robustly decidable? Current Trends in Theoretical Computer Science, 393–408 (2001)
9. Grädel, E., Otto, M., Rosen, E.: Undecidability results on two-variable logics. Arch. Math. Log. 38(4-5), 313–354 (1999)
10. Grädel, E., Walukiewicz, I.: Guarded fixed point logic. In: Proc. LICS 1999. IEEE, Los Alamitos (1999)
11. Libkin, L.: Elements of Finite Model Theory. Springer, Heidelberg (2004)
12. Mortimer, M.: On languages with two variables. Zeitschrift für mathematische Logik und Grundlagen der Mathematik 21(8), 135–140 (1975)
13. ten Cate, B., Segoufin, L.: Unary negation. In: Proc. STACS 2011 (2011)
14. Vardi, M.Y.: Why is modal logic so robustly decidable?. In: Descriptive Complexity and Finite Models, pp. 149–184 (1996)

Locality of Queries Definable in Invariant First-Order Logic with Arbitrary Built-in Predicates

Matthew Anderson[1,⋆], Dieter van Melkebeek[1,⋆],
Nicole Schweikardt[2], and Luc Segoufin[3]

[1] University of Wisconsin - Madison, USA
{mwa,dieter}@cs.wisc.edu
[2] Goethe-Universität Frankfurt am Main, Germany
schweika@informatik.uni-frankfurt.de
[3] INRIA and ENS-Cachan, LSV, France
http://www-rocq.inria.fr/~segoufin

Abstract. We consider first-order formulas over relational structures which may use arbitrary numerical predicates. We require that the validity of the formula is independent of the particular interpretation of the numerical predicates and refer to such formulas as Arb-invariant first-order.

Our main result shows a Gaifman locality theorem: two tuples of a structure with n elements, having the same neighborhood up to distance $(\log n)^{\omega(1)}$, cannot be distinguished by Arb-invariant first-order formulas. When restricting attention to word structures, we can achieve the same quantitative strength for Hanf locality. In both cases we show that our bounds are tight.

Our proof exploits the close connection between Arb-invariant first-order formulas and the complexity class AC^0, and hinges on the tight lower bounds for parity on constant-depth circuits.

1 Introduction

Definability in logics plays an important and delicate role in model checking, a central tool in several areas of computer science such as databases and automated verification. The problem consists in testing whether a given structure satisfies a certain property expressed in the logic. On the one hand, wider expressibility allows for more efficient implementations of a given property. On the other hand, limits on the expressibility keep the model checking task tractable and may be desirable for other reasons. For example, a database can be stored on a disk, which induces a linear order on the elements. An implementation of a query may exploit this order for looping through all the elements of the structure and performing all kinds of numerical computations. At the same time, the result of the query should only depend on the database and not on the linear order that is specific to its current representation on disk. In the database context this requirement is known as the *data independence principle*. In logic we usually speak of *closure under isomorphisms*.

In order to capture such requirements, we consider finite relational structures and queries expressed using first-order (FO) formulas which also have access to a binary

⋆ Partially supported by NSF grants 0728809 and 1017597.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 368–379, 2011.

predicate that is always interpreted as a linear order on the domain of the relational structures. We accommodate numerical computations by also allowing arbitrary numerical predicates in the logic. We require that the result of a query not depend on the actual choice of the linear order when all numerical predicates are interpreted consistent with the linear order. We refer to this logic as *Arb-invariant* FO. The special case where the query does not use any numerical predicate except for the linear order coincides with the well-known notion of *order-invariant* FO (cf., e.g., [12]).

In terms of computational power, Arb-invariant FO expresses precisely the properties computable within the complexity class AC^0, and when restricting to formulas using only the numerical predicates to $+$ and $*$ it corresponds to the uniform version of AC^0 [11]. In particular, Arb-invariant FO is for AC^0 what Arb-invariant Least Fixed Point logic (LFP) is for $\mathrm{P}/poly$ [14], and $(+,*)$-invariant FO is for uniform AC^0 what order-invariant (LFP) is for PTime [16,10]. Note that $+$ and $*$ are definable in order-invariant LFP and therefore can be omitted from the syntax, but this is no longer the case when considering first-order logic. It should be noted, however, that Arb-invariant FO and order-invariant LFP are "logical systems" rather than "logics" in the strict formal sense, as their syntax is undecidable (cf., e.g., [12]).

In this paper we study the expressive power of Arb-invariant FO and therefore the power of the complexity class AC^0. More precisely, we investigate the locality of Arb-invariant FO queries. Locality is a central notion in the study of first-order formulas. It provides good intuition for the expressive power of such formulas, and a powerful tool for showing inexpressibility. For instance, any non-local property such as acyclicity, connectivity, or k-colorability can immediately be shown non-expressible in a logic that exposes a certain amount of locality (see, e.g., [12, Chapter 4]). Locality is also exploited in an essential way in the design of efficient algorithms for evaluating first-order definable queries on certain classes of structures [3,5].

There are two important notions of locality, known as *Gaifman locality* and *Hanf locality*. Both are based on the distance measure on the elements of a structure when viewed as the vertices of the structure's Gaifman graph (in which two elements are connected by an edge whenever they appear together in a tuple of one of the structure's relations). In a nutshell, Gaifman locality means that a query cannot distinguish between two tuples having the same neighborhood type in a given structure, while Hanf locality means that a query cannot distinguish between two structures having the same (multi-)set of neighborhood types. Here, the neighborhood type of a tuple refers to the isomorphism type of the substructure induced by the elements up to distance r from the tuple, where r is a parameter. It is known that Hanf locality implies Gaifman locality, modulo a constant factor loss in the distance parameter r (cf., e.g., [12, Theorem 4.11]).

A well-known result (see, e.g., [12]) shows that FO enjoys both Hanf and Gaifman locality with a "constant" parameter r, i.e. depending only on the query. In the sequel we refer to this property as $\omega(1)$-locality. In the presence of an extra linear order that is part of the structure, all neighborhoods of positive radius degenerate to the entire domain, so all queries are trivially 1-local. Locality becomes meaningful again in order-invariant FO, where the formulas can make use of an order, but the structure does not contain the order and the semantics are independent of the order. It is shown in [8] that order-invariant FO queries are Gaifman $\omega(1)$-local. When we allow arbitrary numerical

predicates, $\omega(1)$-locality turns out to fail, even if we require Arb-invariance. However, by allowing the parameter r to depend on the number n of elements of the structure, we provide an essentially complete picture in the case of Gaifman locality.

Theorem 1. *Arb-invariant* FO *formulas are Gaifman* $(\log n)^{\omega(1)}$*-local, and for every* $c \in \mathbb{N}$ *there exists an Arb-invariant* FO *formula that is not Gaifman* $(\log n)^c$*-local.*

The upper bound in Theorem 1 means that for any query in Arb-invariant FO and any large enough number n, if a structure has n elements and if two tuples of that structure have the same neighborhood up to distance $(\log n)^{f(n)}$ for any function $f \in \omega(1)$, then they cannot be distinguished by the query.

As in the case of order-invariant FO ([8]), the Hanf locality of Arb-invariant FO queries is still open in general. However if we restrict our attention to structures that represent strings, we can establish Hanf locality with the same bounds as in Theorem 1. Recall that order-invariant FO is known to be Hanf $\omega(1)$-local over strings [2]. In the following statement, Arb-invariant FO($Succ$) refers to Arb-invariant queries over string structures.

Theorem 2. *Arb-invariant* FO($Succ$) *sentences are Hanf* $(\log n)^{\omega(1)}$*-local, and for every* $c \in \mathbb{N}$ *there exists an Arb-invariant* FO($Succ$) *sentence that is not Hanf* $(\log n)^c$*-local.*

Proof Techniques. The proof of the upper bound on Gaifman locality in Theorem 1 exploits the tight connection between Arb-invariant FO formulas and the complexity class AC^0. The notion of locality in logic has a similar flavor to the notion of sensitivity in circuit complexity, and AC^0 is known to have low (polylogarithmic) sensitivity [13]. The latter result is closely related to the exponential lower bounds for parity on constant-depth circuits [9]. Rather than going through sensitivity, our argument directly uses the circuit lower bounds, namely as follows.

Given an Arb-invariant FO formula φ that distinguishes two points of the universe whose neighborhoods are of the same type up to distance r, we construct a circuit on $2m = \Theta(r)$ inputs that distinguishes inputs with exactly m ones from inputs with exactly $m + 1$ ones. In the special case of disjoint neighborhoods the circuit actually computes parity. The depth of the circuit is a constant depending on φ, and its size is polynomial in n. The known exponential circuit lower bounds then imply that r is bounded by a polylogarithmic function in n. This argument establishes the upper bound in Theorem 1 for the case of formulas with a single free variable and has some similarities with the proof of [8] establishing the $\omega(1)$-locality of order-invariant FO. However our proof is technically simpler and hinges on circuit lower bounds while the argument in [8] refers to Ehrenfeucht-Fraïssé games.

In order to handle an arbitrary number k of free variables, we follow again the same outline as [8]: we show how to reduce any case with $k > 1$ free variables to one with fewer variables. Our reduction is conceptually harder than the one in [8]. Indeed the reduction in [8] changes the size of the universe which can be done while preserving order-invariance but makes the preservation of Arb-invariance impossible.

The proof of the upper bound in Theorem 2 follows from a reduction to the upper bound in Theorem 1. This strategy differs from the one used in [2], which argues that the expressive power of order-invariant FO on strings is the same as FO (and is hence Hanf $\omega(1)$-local), because Arb-invariant FO($Succ$) can express non-FO properties.

The lower bounds in Theorems 1 and 2 follow because arithmetic predicates like addition and multiplication allow one to define a bijection between the elements of a first-order definable subset S of the domain of polylogarithmic size and an initial segment of the natural numbers [4]. Thus, (the binary representation of) a single element of the entire domain can be used to represent a list of elements of S. By exploiting this, Arb-invariant FO can express, e.g., reachability between two nodes in S by a path of polylogarithmic length.

We defer all proofs to the full paper, which may be found on the authors' websites.

2 Preliminaries

Arb-invariant First-Order Logic. A *relational schema* is a set of relation symbols each with an associated arity. A τ*-structure* M over a relational schema τ is a *finite* set $dom(M)$, the *domain*, containing all the *elements* of M, together with an interpretation R^M of each relation symbol $R \in \tau$. If U is a set of elements of M, then $M_{|U}$ denotes the *induced substructure of* M *on* U. That is, $M_{|U}$ is the structure whose domain is U and whose relations are the relations of M restricted to those tuples containing only elements in U.

We say that two τ-structures M and M' are *isomorphic*, $M \cong M'$, if there exists a bijection $\pi : dom(M) \to dom(M')$ such that for each k-ary relation symbol $R \in \tau$, $(a_1, a_2, \ldots, a_k) \in R^M$ iff $(\pi(a_1), \pi(a_2), \ldots, \pi(a_k)) \in R^{M'}$. We write $\pi : M \cong M'$ to indicate that π is an isomorphism that maps M to M'. If $\boldsymbol{a}$ and $\boldsymbol{b}$ are tuples (of the same length) of distinguished elements of $dom(M)$ and $dom(M')$, respectively, then we write $(M, \boldsymbol{a}) \cong (M', \boldsymbol{b})$ to indicate that there is an isomorphism $\pi : M \cong M'$ which maps $\boldsymbol{a}$ to $\boldsymbol{b}$. All classes of structures considered in this paper are closed under isomorphisms.

Fix an infinite schema σ_{arb}, containing a binary symbol $<$ together with a symbol for each numerical predicate. For instance σ_{arb} contains a symbol $+$ for addition, $*$ for multiplication, and so on. Each numerical predicate is implicitly associated, for every $n \in \mathbb{N}$, with a specific interpretation as a relation of the appropriate arity over the domain $[n] := \{1, 2, \ldots, n\}$. For instance $+$ is associated with the restriction on $[n]$ of the classical relation of addition over $\mathbb{N}$. Reciprocally for each such family of relations, σ_{arb} contains an associated predicate symbol.

Let M be a τ-structure and $n = |dom(M)|$. An Arb-expansion of M is a structure M' over the schema consisting of the disjoint union of τ and σ_{arb} such that $dom(M) = dom(M')$, M and M' agree on all relations in τ, and $<$ is interpreted as a linear order over $dom(M)$. This interpretation induces a bijection between $dom(M)$ and $[n]$, identifying each element of M' with its index relative to $<$. All the numerical predicates are then interpreted over $dom(M')$ via this bijection and their associated interpretation over $[n]$. For instance, $+$ is the ternary relation containing all tuples (a, b, c) of $dom(M')^3$ such that $i + j = k$, where a, b, and c are respectively the i^{th}, j^{th} and k^{th} elements of $dom(M')$ relative to $<$. Note that M' is completely determined by M and the choice of the linear order $<$ on $dom(M)$.

We denote by FO(τ) the first-order logic with respect to the schema τ. We use the standard syntax and semantics for FO (cf., e.g., [12]). If ϕ is a formula, we write $\phi(\boldsymbol{x})$ to denote that $\boldsymbol{x}$ is a list of the free variables of ϕ. We write $(M, \boldsymbol{a})$ when we want to

emphasize the fact that $\boldsymbol{a}$ are distinguished elements of M. We also write $M \models \phi(\boldsymbol{a})$ or $(M, \boldsymbol{a}) \models \phi(\boldsymbol{x})$ to express that the tuple $\boldsymbol{a}$ of elements in $dom(M)$ makes the formula $\phi(\boldsymbol{x})$ true on M.

We denote by $\mathrm{FO}(\tau, \mathrm{Arb})$ the set of first-order formulas using the schema $\tau \cup \sigma_{\mathrm{arb}}$. A formula $\phi(\boldsymbol{x})$ of $\mathrm{FO}(\tau, \mathrm{Arb})$ is said to be *Arb-invariant on a finite structure* M over the schema τ, if for any tuple $\boldsymbol{a}$ of elements of $dom(M)$, and any two Arb-expansions M' and M'' of M we have

$$M' \models \phi(\boldsymbol{a}) \iff M'' \models \phi(\boldsymbol{a}). \tag{1}$$

When $\phi(\boldsymbol{x})$ is Arb-invariant with respect to all finite structures M over a schema, we simply say that $\phi(\boldsymbol{x})$ is *Arb-invariant.*

When $\phi(\boldsymbol{x})$ is an Arb-invariant formula of $\mathrm{FO}(\tau, \mathrm{Arb})$ on M, we write $M \models \phi(\boldsymbol{a})$ whenever there is an Arb-expansion M' of M such that $M' \models \phi(\boldsymbol{a})$. Hence Arb-invariant formulas can be viewed as formulas over τ-structures. We denote by Arb-invariant $\mathrm{FO}(\tau)$ the set of Arb-invariant formulas of $\mathrm{FO}(\tau, \mathrm{Arb})$, or simply Arb-invariant FO if τ is clear from the context. When the formula uses only the predicate $<$ of σ_{arb}, we have the classical notion of order-invariant FO (see [8] and [12]).

Locality. To each structure M we associate an undirected graph $G(M)$, known as the *Gaifman graph* of M, whose vertices are the elements of the domain of M and whose edges relate two elements of M whenever there exists a tuple in one of the relations of M in which both appear. For example, consider a relational schema τ consisting of one binary relation symbol E. Each τ-structure M is then a directed graph in the standard sense, and $G(M)$ coincides with M when ignoring the orientation. Given two elements u and v of a structure M, we denote as $dist^M(u, v)$ the distance between u and v in M which is defined as their distance in the Gaifman graph $G(M)$. If $\boldsymbol{a}$ and $\boldsymbol{b}$ are tuples of elements of M, then $dist^M(\boldsymbol{a}, \boldsymbol{b})$ denotes the minimum distance between any pair of elements (one from $\boldsymbol{a}$ and one from $\boldsymbol{b}$).

For every $r \in \mathbb{N}$ and tuple $\boldsymbol{a} \in dom(M)^k$, the *$r$-ball around $\boldsymbol{a}$ in M* is the set

$$N_r^M(\boldsymbol{a}) \;:=\; \{v \in dom(M) : dist^M(\boldsymbol{a}, v) \leq r\}.$$

and the *r-neighborhood around $\boldsymbol{a}$ in M* is the structure

$$\mathcal{N}_r^M(\boldsymbol{a}) \;:=\; \left(M_{|N_r^M(\boldsymbol{a})}\,,\, \boldsymbol{a}\right).$$

$\mathcal{N}_r^M(\boldsymbol{a})$ is the induced substructure of M on $N_r^M(\boldsymbol{a})$ with k distinguished elements $\boldsymbol{a}$.

Gaifman Locality. Let f be a function from $\mathbb{N}$ to $\mathbb{R}_{\geq 0}$. A formula $\phi(\boldsymbol{x})$ is said to be Gaifman f-local with respect to an infinite class of structures $\mathcal{M}$ if there exists $n_0 \in \mathbb{N}$ such that for any $n > n_0$, for any structure $M \in \mathcal{M}$ with $n = |dom(M)|$, and any tuples $\boldsymbol{a}$ and $\boldsymbol{b}$ we have

$$\mathcal{N}_{f(n)}^M(\boldsymbol{a}) \cong \mathcal{N}_{f(n)}^M(\boldsymbol{b}) \implies M \models \phi(\boldsymbol{a}) \text{ iff } M \models \phi(\boldsymbol{b}). \tag{2}$$

For a set of functions F, a formula is said to be Gaifman F-local if it is Gaifman f-local for every $f \in F$.

Hanf Locality. Let f be a function from $\mathbb{N}$ to $\mathbb{R}_{\geq 0}$. For any two τ-structures M, M' with domain size n we write $M \equiv_{f(n)} M'$ if there is a bijection $h : dom(M) \to dom(M')$ such that for all elements a in the domain of M we have $\mathcal{N}^M_{f(n)}(a) \cong \mathcal{N}^{M'}_{f(n)}(h(a))$.

A sentence ϕ is said to be Hanf f-local if there is a n_0 such that for all τ-structures M, M' with domain size $n > n_0$ we have

$$M \equiv_{f(n)} M' \implies M \models \phi \text{ iff } M' \models \phi. \tag{3}$$

For a set of functions F, a sentence is said to be Hanf F-local if it is Hanf f-local for every $f \in F$.

Circuit complexity. We assume basic familiarity with Boolean circuits. A *family of circuits* is a sequence $(C_m)_{m\in\mathbb{N}}$ such that for all $m \in \mathbb{N}$, C_m is a circuit using m input variables, hence defining a function from $\{0,1\}^m$ to $\{0,1\}$. We say that a language $L \subseteq \{0,1\}^*$ is accepted by a family of circuits $(C_m)_{m\in\mathbb{N}}$ if for all $m \in \mathbb{N}$ and for all binary words w of length m, $C_m(w) = 1$ iff $w \in L$.

When dealing with structures as inputs we need to encode the structures as strings. The precise encoding is not relevant for us as long as it is generic enough. We denote by $\mathrm{Rep}(M)$ the set of all binary encodings of M. Similarly, if $\boldsymbol{a}$ is a tuple of distinguished elements of M, then $\mathrm{Rep}(M, \boldsymbol{a})$ denotes the set of all binary encodings of $(M, \boldsymbol{a})$.

AC^0 *and* $\mathrm{FO}(\tau, Arb)$. A language L is in (nonuniform) AC^0 if there exists a family of circuits $(C_m)_{m\in\mathbb{N}}$ accepting L, a constant $d \in \mathbb{N}$, and a polynomial function $p(m)$ such that for all $m \in \mathbb{N}$ each circuit C_m has depth d and size at most $p(m)$. There is a strong connection between AC^0 and $\mathrm{FO}(\tau, \mathrm{Arb})$ [11]. We make use of the following characterization with respect to Arb-invariant $\mathrm{FO}(\tau, \mathrm{Arb})$.

Lemma 3 (Implicit in [11]). *Let $\phi(\boldsymbol{x})$ be a k-ary $\mathrm{FO}(\tau, Arb)$ formula which is Arb-invariant on a class of τ-structures $\mathcal{M}$. There exists a family of constant-depth and polynomial-size circuits $(C_m)_{m\in\mathbb{N}}$ such that for each $M \in \mathcal{M}$, $\boldsymbol{a} \in dom(M)^k$, and $\Gamma \in \mathrm{Rep}(M, \boldsymbol{a})$,*

$$C_{|\Gamma|}(\Gamma) = 1 \iff M \models \phi(\boldsymbol{a}).$$

Lower bounds. Our locality bounds hinge on the well-known exponential size lower bounds for constant-depth circuits that compute parity [1,6,17,9]. In fact, we use the following somewhat stronger promise version. For a binary word $w \in \{0,1\}^*$, we let $|w|_1$ denote the number of $1s$ in w.

Lemma 4 (Implicit in [9, Theorem 5.1]). *For any $d \in \mathbb{N}$, there are constants c and m_0 such that for $m > m_0$ there is no circuit of depth d and size $2^{cm^{\frac{1}{d-1}}}$ that $w \in \{0,1\}^{2m}$ accepts all inputs $w \in \{0,1\}^{2m}$ with $|w|_1 = m$ and rejects all inputs with $|w|_1 = m+1$.*

3 Gaifman Locality

We now prove the main result of the paper – the upper bound in Theorem 1. Recall, our theorem claims that every Arb-invariant FO formula is Gaifman $(\log n)^{\omega(1)}$-local. In fact we prove the following slightly stronger version.

Theorem 5. *For any* FO(τ, Arb) *formula* $\phi(\boldsymbol{x})$*, and infinite class* $\mathcal{M}$ *of* τ*-structures, if* $\phi(\boldsymbol{x})$ *is Arb-invariant on* $\mathcal{M}$*, then* $\phi(\boldsymbol{x})$ *is Gaifman* $(\log n)^{\omega(1)}$*-local on* $\mathcal{M}$*.*

We now briefly sketch the overall proof of Theorem 5. Suppose we have two tuples, $\boldsymbol{a}$ and $\boldsymbol{b}$, on a τ-structure M, with domain size n, such that their r-neighborhoods, $\mathcal{N}_r^M(\boldsymbol{a})$ and $\mathcal{N}_r^M(\boldsymbol{b})$, are isomorphic (for some big enough r). Suppose that there is a FO(τ, Arb) formula $\phi(\boldsymbol{x})$ which is able to distinguish between $\boldsymbol{a}$ and $\boldsymbol{b}$ on M while being Arb-invariant on M. Using the link between Arb-invariant FO(τ, Arb) formulas and AC^0 circuits from Lemma 3, we can view the formula $\phi(\boldsymbol{x})$ as a constant-depth circuit C.

We are able to show that because $\phi(\boldsymbol{x})$ is Arb-invariant and distinguishes between $\boldsymbol{a}$ and $\boldsymbol{b}$ on M, we can construct from the circuit C and structure M another circuit $\tilde{C}$ that for $(2m)$-length binary strings w distinguishes between the cases when w contains m occurrences of 1 and $m+1$ occurrences, for some m depending on r. This is the *key step* in our argument. If this happens for infinitely many n, we get a family of circuits computing the promise problem described in Lemma 4. We can argue that the size of $\tilde{C}$ is polynomial in n and the depth of $\tilde{C}$ only depends on $\phi(\boldsymbol{x})$ and hence is constant. Therefore if $m \in (\log n)^{\omega(1)}$ the family of circuits $\tilde{C}$ we construct violates Lemma 4 hence $\phi(\boldsymbol{x})$ cannot distinguish between tuples which have isomorphic r-neighborhoods. Our construction is such that m is linearly related to r and therefore $\phi(\boldsymbol{x})$ is Gaifman $(\log n)^{\omega(1)}$-local.

3.1 Unary Formulas

In this subsection we consider only unary FO formulas $\phi(x)$. For didactic reasons we first assume that the r-neighborhoods of the elements a and b are disjoint. We argue that we can perform the key step in this setting, then consider the general unary case. We conclude by arguing a unary version of Theorem 5.

For clarity we describe the intuition with respect to structures that are graphs. Let M be a graph $G = (V, E)$ and take two vertices $a, b \in V$ such that $\pi : \mathcal{N}_r^G(a) \cong \mathcal{N}_r^G(b)$. Suppose, for the sake of contradiction, that there is a unary FO formula $\phi(x)$, which is Arb-invariant on G, such that $G \models \phi(a) \wedge \neg\phi(b)$. Applying Lemma 3 to ϕ gives us a circuit C which, for any vertex $c \in V$, outputs the same value for all strings in $\text{Rep}(G, c)$, and distinguishes $\text{Rep}(G, a)$ from $\text{Rep}(G, b)$.

Disjoint neighborhoods. Let us assume that $\mathcal{N}_r^G(a) \cap \mathcal{N}_r^G(b) = \emptyset$. In this setting it turns out we can pick $r = m$. The neighborhood isomorphism, $\pi : \mathcal{N}_r^G(a) \cong \mathcal{N}_r^G(b)$, implies that the balls of radius $i < r$ around a and b are isomorphic and disjoint in G. Consider the following procedure, depicted in Figure 1. For some $i \in [m]$ cut all the edges linking nodes at distance $i-1$ from a or b to nodes at distance i. Now, swap the positions of the $(i-1)$-neighborhoods around a and b and reconnect the edges in a way that respects the isomorphism π. The resulting graph is isomorphic to G, but the relative positions of a and b have swapped.

Using this intuition we construct a new graph G_w from G, a, and b that depends on a sequence of m Boolean variables $w := w_1 w_2 \cdots w_m$. We construct G_w so that for each variable w_i, we swap the relative positions of the radius $i-1$ balls around a and b iff

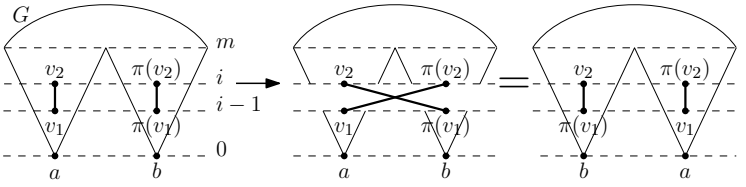


Fig. 1. Diagram for swapping the neighborhoods of a and b of radius i, conditioned on $w_i = 1$

w_i is 1. The number of such swaps is $|w|_1$. The m-neighborhood isomorphism between a and b implies that $G_w \cong G$. When $|w|_1$ is even $(G_w, a) \cong (G, a)$ and when $|w|_1$ is odd $(G_w, a) \cong (G, b)$.

Using the above construction of G_w we derive a circuit $\tilde{C}$ from C that computes parity on m bits. The circuit $\tilde{C}$ first computes a representation $\Gamma_w \in \mathrm{Rep}(G_w, a)$, and then simulates C on input Γ_w. The above distinguishing property then implies that $\tilde{C}$ computes parity on m bits. To construct Γ_w we start with a fixed string in $\mathrm{Rep}(G, a)$ and transform it into an element of $\mathrm{Rep}(G_w, a)$ by modifying the edges to switch between the shells in the manner suggested above. Observe that the presence of each edge in G_w depends on at most a single bit of w. This property implies that Γ_w consists of constants and variables in w or their negations. This means that $\tilde{C}$ is no larger or deeper than C.

We formalize this intuition for general structures and obtain the following lemma.

Lemma 6. *Let $m \in \mathbb{N}$. Let M be a structure. Let $a, b \in dom(M)$ such that $dist^M(a, b) > 2m$ and $\mathcal{N}_m^M(a) \cong \mathcal{N}_m^M(b)$. Let C be a circuit that accepts all strings in $\mathrm{Rep}(M, a)$, and rejects all strings in $\mathrm{Rep}(M, b)$. There is a circuit $\tilde{C}$ with the same size and depth as C that computes parity on m bits.*

General Unary Case. We now develop the transformation corresponding to Lemma 6 for the general unary case, where the r-neighborhoods around a and b may overlap. As before, we describe the intuition in terms of structures that are graphs.

Consider the iterative application of the isomorphism π to a. We distinguish between two cases. The first case occurs when this iteration travels far from a. That is, for some $t \in \mathbb{N}$, $\pi^t(a)$ is a point c that is far from a. We choose r large enough so that a large neighborhood around c is isomorphic to the neighborhood around a or b. By the triangle inequality, since a is far from c either: b is far from a, or c is far from a and b. In the latter case C must distinguish between either a and c or b and c. Therefore, w.l.o.g., we have a pair of vertices that are distinguished by C, and whose neighborhoods are isomorphic and disjoint. For this pair of vertices, we are in the disjoint case and Lemma 6 can be applied to produce a small circuit that computes parity.

The other case occurs when the iterative application of π to a stays close to a (and b). Let S_0 be the orbit of a under π, and let S_i be the vertices at distance i from S_0, for $i \in [2m]$. Because $\pi(S_0) = S_0$ and π is a partial isomorphism on G, the shells S_i are closed under π.

We now play a game similar to the disjoint case. Consider the following procedure, depicted in Figure 2. For some $i \in [2m]$ cut all edges between the shell S_{i-1} and S_i. "Rotate" the radius $i - 1$ ball around S_0 by π relative to S_i, and reconnect the edges.

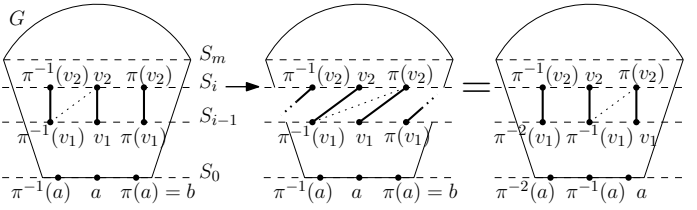


Fig. 2. Diagram for rotating the shell of radius i around S_0 when $w_i = 1$

Because the shells are closed under π, the resulting graph is isomorphic to G. Further, the positions of a and b have shifted relative to an application of π.

As before, we encode this behavior into a modified graph G_w depending on a sequence of $2m$ Boolean variables $w := w_1 w_2 \cdots w_{2m}$. When $w_i = 0$, we preserve the edges between the shells S_{i-1} and S_i. When $w_i = 1$ we rotate the edges by π. That is, an edge $(v_1, v_2) \in (S_{i-1} \times S_i) \cap E$ becomes the edge $(v_1, \pi(v_2))$ in G_w. The neighborhood isomorphism between a and b implies that $G \cong G_w$. We can argue that

$$(G_w, a) \cong (G, \pi^{|w|_1}(a)). \tag{4}$$

We define the circuit $\tilde{C}$ to simulate C on an input $\Gamma_w \in \mathrm{Rep}(G_w, a)$. The above distinguishing property implies that $\tilde{C}$ distinguishes between $|w|_1 \equiv 0 \bmod |S_0|$ and $|w|_1 \equiv 1 \bmod |S_0|$. This is not quite the promise problem defined in Lemma 4. For this reason we modify the construction to shift a by m applications of π^{-1} in Γ_w. This means that $\Gamma_w \in \mathrm{Rep}(G_w, \pi^{-m}(a))$ and $\tilde{C}$ can distinguish between $|w|_1 \equiv m \bmod |S_0|$ and $|w|_1 \equiv m + 1 \bmod |S_0|$. This is ruled out by Lemma 4, completing the argument.

For general structures, the idea is formalized in the following lemma.

Lemma 7. *Let $m \in \mathbb{N}$. Let M be a structure. Let $a, b \in dom(M)$ such that $\mathcal{N}^M_{12m}(a) \cong \mathcal{N}^M_{12m}(b)$. Let C be a circuit that accepts all strings in $\mathrm{Rep}(M, a)$ and rejects all strings in $\mathrm{Rep}(M, b)$, and for each $c \in dom(M)$, C has the same output for each string in $\mathrm{Rep}(M, c)$. There is a circuit $\tilde{C}$ with the same size and depth as C that distinguishes $|w|_1 = m$ and $|w|_1 = m + 1$ for $w \in \{0,1\}^{2m}$.*

Proof of Theorem 5 in the case of unary formulas. Now that we know how to construct the circuit $\tilde{C}$ as in Lemmas 6 and 7, we are ready to finish the proof of Theorem 5 in the unary case. Assume that $\phi(x)$ is a unary formula of $\mathrm{FO}(\tau, \mathrm{Arb})$ that is Arb-invariant with respect to an infinite class of τ-structures $\mathcal{M}$.

Since $\phi(x)$ is $\mathrm{FO}(\tau, \mathrm{Arb})$, it is computable by a family of circuits in AC^0 (cf., Lemma 3). That is, there are a constant d, polynomials $s(n)$ and $r(n)$, and a circuit family $(C_{r(n)})_{n \in \mathbb{N}}$ with depth d and size $s(n)$ such that, for every $n \in \mathbb{N}$, the circuit $C_{r(n)}$ computes $\phi(x)$ on structures $M \in \mathcal{M}$ with $|dom(M)| = n$ and $r(n)$ bounds the length of the binary encoding of (M, a) for any $a \in dom(M)$. Since $\phi(x)$ is Arb-invariant on M, $C_{r(n)}$ has the same output for all strings in $\mathrm{Rep}(M, a)$ for each $a \in dom(M)$.

Now, for the sake of contradiction, suppose $\phi(x)$ is *not* Gaifman $(\log n)^{\omega(1)}$-local on $\mathcal{M}$. This implies that there is an infinite subclass of structures $\mathcal{M}' \subseteq \mathcal{M}$ and a function

$f(n)$ in $(\log n)^{\omega(1)}$, where for each $M \in \mathcal{M}'$, $\phi(x)$ distinguishes between two elements $a, b \in dom(M)$ having isomorphic $f(n)$-neighborhoods.

Consider some structure $M \in \mathcal{M}'$, with $n = |dom(M)|$. Let $m := \lfloor \frac{f(n)}{12} \rfloor$. By the above, there exists $a, b \in dom(M)$ such that $\mathcal{N}^M_{12m}(a) \cong \mathcal{N}^M_{12m}(b)$ and $M \models \phi(a) \wedge \neg\phi(b)$ without loss of generality. Let $C := C_{r(n)}$; this circuit then satisfies the assumptions of Lemma 7. From the lemma, we obtain a circuit $\tilde{C}$ of depth d and size $s(n)$ that distinguishes between $|w|_1 = m$ and $|w|_1 = m + 1$ for $w \in \{0,1\}^{2m}$.

From Lemma 4 we obtain that $s(n) > 2^{cm^{1/(d-1)}}$. Noting that $m = (\log n)^{\omega(1)}$, s is polynomial, and d is constant, this yields a contradiction by choosing $M \in \mathcal{M}'$ sufficiently large, completing the proof. □

3.2 k-ary Formulas

To argue Theorem 5 in the case of formulas with an arbitrary number of free variables, we prove the following reduction. Given a k-ary FO(Arb) formula ϕ that is Arb-invariant on the structure M and distinguishes two k-tuples $\boldsymbol{a}$ and $\boldsymbol{b}$ that have isomorphic r-neighborhoods, we produce, for some $k' < k$, a k'-ary formula FO(Arb) ϕ' that is Arb-invariant on an extended structure M' and distinguishes between two k'-tuples $\boldsymbol{a}'$ and $\boldsymbol{b}'$ that have isomorphic r'-neighborhoods. Furthermore, r' is only slightly smaller than r. The formal statement of the reduction is as follows.

Lemma 8. *Let $k, d \in \mathbb{N}$, r a function from $\mathbb{N}$ to $\mathbb{R}_{\geq 0}$, and τ be a schema. Fix $\alpha \leq \frac{1}{7k}$. Let $\phi(\boldsymbol{x})$ be a k-ary* FO(τ, Arb) *formula of quantifier-depth d that is Arb-invariant over an infinite class of τ-structures $\mathcal{M}$ and that is not Gaifman r-local.*

Then there is $k' < k$, a schema $\tau' \supseteq \tau$, an infinite class of τ'-structures $\mathcal{M}'$ and a k'-ary FO(τ', Arb) *formula $\phi'(\boldsymbol{y})$ of quantifier-depth $(d+(k-k'))$ that is Arb-invariant over $\mathcal{M}'$ and not Gaifman αr-local.*

Repeated application of this lemma transforms a distinguishing k-ary formula into a distinguishing unary formula with slightly weaker parameters. For large enough initial radius this is sufficient to contradict the Gaifman locality of unary formulas.

4 Hanf Locality

The main result of this section is the upper bound in Theorem 2, which states that Arb-invariant FO formulas over strings are Hanf $(\log n)^{\omega(1)}$-local.

Fix a finite alphabet $\mathbb{A}$ and consider structures over the schema τ_s containing one unary predicate per element of $\mathbb{A}$ and one binary predicate E. Let $\mathcal{S}$ be the class of τ_s-structures M that interprets E as a successor relation and where each element of M belongs to exactly one of the unary predicates in τ_s. Each structure in $\mathcal{S}$ represents a string in the obvious way and we blur the distinction between a string w and its actual representation as a structure. We then consider FO$(\tau_s \cup \sigma_{\text{arb}})$ formulas that are Arb-invariant over $\mathcal{S}$ and denote the corresponding set of formulas by Arb-invariant FO$(Succ)$. We say that a language $L \subseteq \mathbb{A}^*$ is definable in Arb-invariant FO$(Succ)$ if there is a sentence of Arb-invariant FO$(Succ)$ whose set of models in $\mathcal{S}$ is exactly L.

The proof of the upper bound in Theorem 2 has several steps. We first introduce a closure property of languages allowing, under certain conditions, substrings inside a word to be swapped without affecting language membership. We then argue that languages definable in Arb-invariant FO(*Succ*) have this closure property. Finally, we conclude by proving that this closure property implies that Arb-invariant FO(*Succ*) sentences are Hanf $(\log n)^{\omega(1)}$-local. In the following, we describe these steps in some more detail.

Arb-invariant FO(*Succ*) is Closed Under Swaps. Let $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$. A language L is said to be *closed under $f(n)$-swaps* if there exists a $n_0 \in \mathbb{N}$ such that for all strings $w := xuyvz \in \mathbb{A}^*$, with $|w| = n > n_0$, and where i, j, i', and j' are, respectively, the positions in w immediately before the substrings u, y, v, and z, and $\mathcal{N}^w_{f(n)}(i) \cong \mathcal{N}^w_{f(n)}(i')$ and $\mathcal{N}^w_{f(n)}(j) \cong \mathcal{N}^w_{f(n)}(j')$ we have: $w := xuyvz \in L$ iff $w' := xvyuz \in L$. A language is closed under F-swaps if is it closed under $f(n)$-swaps for all $f \in F$.

Let ϕ be an Arb-invariant FO(*Succ*) sentence. Suppose the strings w and w' satisfy the initial conditions for closure under $(\log n)^{\omega(1)}$-swaps, but are distinguished by ϕ. We first consider the case when the four $f(n)$-neighborhoods of i, j, i', j' are disjoint. In this case not only do the neighborhoods around the strings u and v look same, but u and v are far apart. We define a FO(Arb) formula ψ derived from ϕ and a structure M derived from w and w', such that ψ on M simulates ϕ on either w or w' depending on the input to ψ. Moreover, ψ is Arb-invariant on M and the input tuples that ψ distinguishes have large isomorphic neighborhoods, implied by the neighborhood isomorphisms among i, j, i', and j'. Applying our Gaifman locality theorem (Theorem 1) to the formula ψ induces a contradiction.

When the neighborhoods are not disjoint we reduce to the disjoint case by making two key observations. The first is that when some of the neighborhoods overlap only a small amount there is freedom to select slightly smaller neighborhoods that are pairwise disjoint, though still induce the same swap. The second insight is that when several of the neighborhoods have considerable overlap, the neighborhood isomorphisms induce periodic behavior within those neighborhoods. So much so that the substrings uyv and vyu must be identical. This contradicts the fact that w and w' are distinct. This intuition is formalized in the following lemma.

Lemma 9. *If L is a language definable in Arb-invariant* FO(*Succ*) *then L is closed under $(\log n)^{\omega(1)}$-swaps.*

Arb-invariant FO(*Succ*) is Hanf $(\log n)^{\omega(1)}$-local. We are now ready to prove the upper bound of Theorem 2. Consider a pair of equal length strings w, w' such that $w \equiv_{f(n)} w'$ for some bijection h. Observe that if $w = w'$, we can choose h to be the identity. The identity mapping is monotone in the sense that for each position $i \in [n]$, for all $j < i$, $h(j) < h(i)$. When $w \neq w'$, h is not the identity and not monotone. However, h is monotone when considering only the first position. We extend the set which h is monotone on by $(\log n)^{\omega(1)}$-swapping substrings of w while being careful to preserve the bijection to w'. Eventually h becomes monotone with respect to all positions and is the identity. The final insight is this, if we only perform $(\log n)^{\omega(1)}$-swaps language membership is maintained by Lemma 9. Thus, we transform between w and w' without changing language membership, so $w \in L$ iff $w' \in L$. Hence Arb-invariant FO(*Succ*) is Hanf $(\log n)^{\omega(1)}$-local.

5 Discussion

We have established the precise level of locality of Arb-invariant FO formulas for the Gaifman notion of locality. We leave it as an open problem whether the same bounds could be achieved for the Hanf notion of locality. We managed to prove Hanf locality for the special case of strings and we believe that a similar argument also works for trees and possibly graphs of bounded treewidth.

As pointed out in [7] "it would be interesting to see a small complexity class like uniform AC^0 [...] can be captured by a logic" (recall from the introduction that although Arb-invariant FO does capture AC^0, it does not have an effective syntax). As a (simple) first step towards a solution to this problem, in the journal version of this paper we will show that *over regular languages*, Arb-invariant FO($Succ$) has exactly the same expressive power as FO($Succ, lm$), where lm is the family of predicates testing the length of a string modulo some fixed number. Note that when combining this result with the one of [15], this shows all the numerical predicates do not bring any extra expressive power than the one of addition over regular languages.

References

1. Ajtai, M.: Σ^1_1-formulae on finite structures. APAL 24(1), 1–48 (1983)
2. Benedikt, M., Segoufin, L.: Towards a characterization of order-invariant queries over tame structures. JSL 74(1), 168–186 (2009)
3. Dawar, A., Grohe, M., Kreutzer, S.: Locally excluding a minor. In: Proc. of LICS (2007)
4. Durand, A., Lautemann, C., More, M.: Counting results in weak formalisms. In: Circuits, Logic, and Games. No. 06451 in Dagstuhl Seminar Proc. (2007)
5. Dvorak, Z., Král, D., Thomas, R.: Deciding first-order properties for sparse graphs. In: Proc. of FOCS (2010)
6. Furst, M., Saxe, J., Sipser, M.: Parity, circuits, and the polynomial-time hierarchy. TOCS 17(1), 13–27 (1984)
7. Grohe, M.: Fixed-point definability and polynomial time. In: Grädel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 20–23. Springer, Heidelberg (2009)
8. Grohe, M., Schwentick, T.: Locality of order-invariant first-order formulas. ACM TOCL 1(1), 112–130 (2000)
9. Håstad, J.: Computational limitations for small-depth circuits. Ph.D. thesis, MIT (1986)
10. Immerman, N.: Relational queries computable in polynomial time. Information and Control 68(1-3), 86–104 (1986)
11. Immerman, N.: Languages that capture complexity classes. SICOMP 16(4), 760–778 (1987)
12. Libkin, L.: Elements of Finite Model Theory. Springer, Heidelberg (2004)
13. Linial, N., Mansour, Y., Nisan, N.: Constant depth circuits, Fourier transform, and learnability. JACM 40(3), 620 (1993)
14. Makowsky, J.A.: Invariant definability and P/*poly*. In: Gottlob, G., Grandjean, E., Seyr, K. (eds.) CSL 1998. LNCS, vol. 1584, pp. 142–158. Springer, Heidelberg (1999)
15. Schweikardt, N., Segoufin, L.: Addition-invariant FO and regularity. In: Proc. of LICS (2010)
16. Vardi, M.: The complexity of relational query languages. In: Proc. of STOC (1982)
17. Yao, A.: Separating the polynomial-time hierarchy by oracles. In: Proc. of FOCS (1985)

Modular Markovian Logic

Luca Cardelli[1], Kim G. Larsen[2], and Radu Mardare[2,⋆]

[1] Microsoft Research, Cambridge, UK
[2] Aalborg University, Denmark

Abstract. We introduce Modular Markovian Logic (MML) for compositional continuous-time and continuous-space Markov processes. MML combines operators specific to stochastic logics with operators reflecting the modular structure of the models, similar to those used by spatial and separation logics. We present a complete Hilbert-style axiomatization for MML, prove the small model property and analyze the relation between stochastic bisimulation and logical equivalence.

1 Introduction

Complex networks (e.g., embedded systems, communication networks, the Internet etc.) and complex systems (e.g., biological, ecological, social, financial, etc.) are often modelled as stochastic processes, to encapsulate a lack of knowledge or inherent randomness. Such systems are frequently modular in nature, consisting of parts which are systems in their own right. Their global behaviour depends on the behaviour of their parts and on the links which connect them. Understanding such systems requires integration of local stochastic information in a formal way, in order to address questions such as: *"to what extent is it possible to derive global properties of the system from the local properties of its modules?"*.

This is a problem of fundamental importance in complex systems that has been usually addressed semantically: probabilistic and stochastic process algebras, for instance, aim at describing compositionally the behaviour of a system from the behaviours of its subsystems taking into account various types of synchronization or communication. This approach is quite restrictive, as process algebras are not logics: one cannot express basic logical operations such as conjunction, disjunction, implication or negation of properties. Usually, to do this, people use logics such as temporal logics, modal μ-calculus [21] or Hennessy-Milner logic [18] to express properties of transition systems. But these are global properties only and no logical framework developed so far allows reasoning on stochastic systems and subsystems at the same time.

In this paper we develop a logical framework called Modular Markovian Logic (MML) that tackles this problem by organizing qualitative and quantitative properties of stochastic systems in hierarchical, modular structures, thereby proving global properties from the local properties of modules. Formally, denoting *"process P has the property ϕ"* by $P \Vdash \phi$ and letting "$\otimes$" be the composition operator, we aim to establish a framework containing modular proof rules of the form $\dfrac{P_1 \Vdash \phi_1, ..., P_k \Vdash \phi_k}{P_1 \otimes ... \otimes P_k \Vdash \rho} C(\rho, \phi_1, .., \phi_k)$, where C is a logical constraint.

⋆ Research partially supported by Sapere Aude: DFF-Young Researchers Grant 10-085054 of the Danish Council for Independent Research.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 380–391, 2011.

To gain this level of expressivity, MML combines stochastic operators similar to the ones of Aumann's system [1,14] with modular operators similar to the ones used in spatial logics [5,6] and in separation logics [30]. For an observable action a and a positive rational r, the operator "L^a_r" of MML expresses the fact that a process can perform an a-transition with the rate[1] at least r. In addition, the *composition operator* "|" joins logical terms and directly expresses properties of the combined subsystems.

On the semantic level, we introduce the *modular Markov processes* (MMPs) which are (continuous-) labelled Markov processes [13,28] enriched with an algebraic structure. This algebra defines the composition of Markovian systems and establishes the relation between a system and its subsystems. The composition of behaviours satisfies a general synchronization pattern which subsumes most of the classical notions of parallel composition found in process algebras.

We define the modular Markovian logic for a semantics based on MMPs. We investigate the relation between stochastic bisimulations of MMPs and logical equivalence induced by MML over the class of MMPs. We present a complete Hilbert style axiomatization of MML for the Markovian semantics and prove the small model property.

Research context. Labelled Markov process (LMPs) are introduced in [12,3,13,28] and they generalize most of the models of Markovian systems. A similar concept, Harsanyi type space (HTS), has been studied in the context of belief systems [15,27]. MMPs are built on top of these, by exploiting their equivalence proved in [10]. In addition, MMPs have inbuilt an algebraic structure that extends, for continuous space and time, the concepts of the Markov chain algebra [4].

Probabilistic logics have been studied for LMPs (probabilistic versions of temporal and Hennessy-Milner logics [13,11,28]), for HTSs (Aumann's system [1,14]) and in a more general context [9]. The first class focuses on model checking and logical characterization of stochastic bisimulation, while for Aumann's system also axiomatization issues have been addressed [17,31]. In [8] we have proposed a completely axiomatized stochastic logic that combines features of the two classes of logics. In this paper we extend the stochastic logic with modular operators that allow us, in addition, to investigate the algebraic structures of the models.

Modular logics, such as spatial logics [5,6] and separation logic [30] have been developed for concurrent nondeterministic systems, but to the best of our knowledge, no stochastic or probabilistic version of these have been studied.

The paper is organized as follows. Section 2 introduces basic concepts used in the paper. Section 3 defines MMPs and their bisimulation. Section 4 presents MML and results concerning the relationship between logical equivalence and bisimulation. Section 5 contains the axiomatic system of MML, the soundness and completeness metatheorems and the small model property. The paper also contains a conclusive section.

2 Preliminary Definitions

In this section we establish the terminology used in the paper.

[1] The rate of a transition is the parameter of an exponentially distributed random variable that characterizes, for Markovian processes, the duration of the transition.

Given a set M, $\Sigma \subseteq 2^M$ that contains M and is closed under complement and countable union is a *σ-algebra* over M; (M, Σ) is a *measurable space* and the elements of Σ are *measurable sets*. $\Omega \subseteq 2^M$ is a *base for* Σ if Σ is the closure of Ω under complement and countable union; we write $\overline{\Omega} = \Sigma$.

A relation $\Re \subseteq M \times M$ is *non-wellfounded* if there exists $\{m_i \in M \mid i \in \mathbb{N}\}$ such that for each $i \in \mathbb{N}$, $(m_i, m_{i+1}) \in \Re$; otherwise it is *wellfounded*. A subset $N \subseteq M$ is $\Re$-closed iff $\{m \in M \mid \exists n \in N, (m, n) \in \Re\} \subseteq N$. If (M, Σ) is a measurable space and $\Re \subseteq M \times M$, $\Sigma(\Re)$ denotes the set of measurable $\Re$-closed subsets of M.

A *measure* on (M, Σ) is a function $\mu : \Sigma \to \mathbb{R}^+$ such that $\mu(\emptyset) = 0$ and for $\{N_i | i \in I \subseteq \mathbb{N}\} \subseteq \Sigma$ with pairwise disjoint elements, $\mu(\bigcup_{i\in I} N_i) = \sum_{i\in I} \mu(N_i)$.

Let $\Delta(M, \Sigma)$ be the class of measures on (M, Σ). We organize it as a measurable space by considering the σ-algebra generated, for arbitrary $S \in \Sigma$ and $r > 0$, by the sets $\{\mu \in \Delta(M, \Sigma) : \mu(S) \geq r\}$.

Given two measurable spaces (M, Σ) and (N, Θ), a mapping $f : M \to N$ is *measurable* if for any $T \in \Theta$, $f^{-1}(T) \in \Sigma$. We use $[\![M \to N]\!]$ to denote the class of measurable mappings from (M, Σ) to (N, Θ).

Central for this paper is the notion of an *analytic set*. We only recall the main definition and mention the properties of analytic sets used in our proofs. For detailed discussion on this topic related to Markov processes, the reader is referred to [28] (Section 7.5) or to [10] (Section 4.4).

A metric space (M, d) is *complete* if every Cauchy sequence converges in M.

A *Polish space* is the topological space underlying a complete metric space with a countable dense subset. Note that any discrete space is Polish.

An *analytic set* is the image of a Polish space under a continuous function between Polish spaces. Note that any Polish space is an analytic set.

There are some basic facts about analytic sets that we use in this paper. Firstly, an analytic set, as measurable space, has a denumerable base with disjoint elements. Secondly, If $\mathcal{M}_1, \mathcal{M}_2$ are analytic sets with Σ_1, Σ_2 the Borel algebras generated by their topologies, then the product space $\mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2$ with the Borel algebra Σ generated by the product topology is an analytic set.

3 Modular Markov Processes

For the beginning we introduce continuous Markov processes (CMPs) for a finite set $\mathcal{A}$ of *actions*. CMPs are coalgebraic structures that encode stochastic behaviors. If m is the current state of the system, N a measurable set of states and $a \in \mathcal{A}$, $\theta(a)(m)$ is a measure on the state space and $\theta(a)(m)(N) \in \mathbb{R}^+$ represents the *rate* of an exponentially distributed random variable that characterizes the duration of an a-transition from m to arbitrary $n \in N$. Indeterminacy is resolved by races between events executing at different rates.

Definition 1 (Continuous Markov processes). *Given an analytic set (M, Σ), where Σ is the Borel algebra generated by the topology, an* $\mathcal{A}$-continuous Markov kernel *is*

a tuple $\mathcal{K} = (M, \Sigma, \theta)$*, where* $\theta : \mathcal{A} \to [\![M \to \Delta(M, \Sigma)]\!]$*. If* $m \in M$*,* $(\mathcal{K}, m)$ *is an* $\mathcal{A}$-continuous Markov process[2].

Let $\mathfrak{K}$ be the class of $\mathcal{A}$-CMKs; $\mathcal{K}, \mathcal{K}_i, \mathcal{K}'$ are used to range over $\mathcal{A}$-CMKs. Stochastic bisimulation follows the line of Larsen-Skou bisimulation [23,11,28].

Definition 2 (Stochastic Bisimilarity). *Given* $\mathcal{K} = (M, \Sigma, \theta) \in \mathfrak{K}$*, a* rate-bisimulation relation *on* $\mathcal{K}$ *is a relation* $\mathfrak{R} \subseteq M \times M$ *such that* $(m, n) \in \mathfrak{R}$ *iff for any* $C \in \Sigma(\mathfrak{R})$ *and any* $a \in \mathcal{A}$*,* $\theta(a)(m)(C) = \theta(a)(n)(C)$*.*
Two processes $(\mathcal{K}, m)$ *and* $(\mathcal{K}, n)$ *are* stochastic bisimilar*, written* $m \sim_{\mathcal{K}} n$*, if they are related by a rate-bisimulation relation.*

Two processes $(\mathcal{K}, m)$ and $(\mathcal{K}', m')$ are *stochastic bisimilar*, written $(\mathcal{K}, m) \sim (\mathcal{K}', m')$, iff $m \sim_{\mathcal{K} \uplus \mathcal{K}'} m'$, where $\mathcal{K} \uplus \mathcal{K}'$ is the disjoint union of $\mathcal{K}$ and $\mathcal{K}'$. We call the relation $\sim$ *stochastic bisimulation*.

3.1 Synchronization

To define the modular Markov processes we need a general notion of synchronization of CMPs that we introduce following the general line of [20]. For this, we assume extra structure on the set $\mathcal{A}$ of actions.

Firstly, we consider a *synchronisation function* $*$ that is a partial function $* : \mathcal{A} \times \mathcal{A} \hookrightarrow \mathcal{A}$ which associates to some $a, b \in \mathcal{A}$ an action $a * b \in \mathcal{A}$ interpreted as the synchronisation of a and b. In this way we can mimic various synchronisation paradigms. For instance, the CCS-style synchronisation [26] requires that $a * \overline{a} = \tau$, where $\tau \in \mathcal{A}$ is a special action; CSP-style [19] requires that $a * a = a$; for interleaving and ACP-style [2] we need to assume that there exists a reflexive transition $\delta \in \mathcal{A}$ such that for any $a \in \mathcal{A}$, $a * \delta = a$. Similarly, most classical notions of parallel composition in process algebras may be expressed by a suitable synchronization function.
The only formal requirement is that $*$, as an operation, is *commutative* ($a * b = b * a$).

Secondly, we assume a function $\bullet : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ that computes, given the rates r and s of the actions a and b respectively, the rate $r \bullet s$ of the synchronisation $a * b$. Examples of such function are the *mass action law* used with stochastic Pi-calculus [29] and other models of bio-chemical interactions and the *minimal rate law* used by PEPA [16] for applications in performance evaluation. The formal requirements are:

$\bullet : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ is a *continuous* function that, as an operation, is *commutative* ($r \bullet s = s \bullet r$), *associative* ($(r \bullet s) \bullet t = r \bullet (s \bullet t)$) and *bilinear* ($(r_1 + r_2) \bullet s = (r_1 \bullet s) + (r_2 \bullet s)$ and $s \bullet (r_1 + r_2) = (s \bullet r_1) + (s \bullet r_2)$).

These two functions define the synchronization of two CMPs as follows.

Definition 3. *For* $i = 1, 2$*, let* $\mathcal{K}_i = (M_i, \Sigma_i, \theta_i) \in \mathfrak{K}$ *and* $\Delta_i \subseteq \Sigma_i$ *denumerable bases with disjoint elements.* $\mathcal{K} = (M, \Sigma, \theta)$ *is the* product *of* $\mathcal{K}_1$ *and* $\mathcal{K}_2$*, written* $\mathcal{K} = \mathcal{K}_1 \times \mathcal{K}_2$*, if* $M = M_1 \times M_2$*,* $\Sigma = \overline{\Delta_1 \times \Delta_2}$ *and* $\theta : \mathcal{A} \to [M \to [\Sigma \to \mathbb{R}^+]]$ *is defined, for* $m_i \in M_i$*,* $a \in \mathcal{A}$ *and* $S = \bigcup_{k \in K \subseteq \mathbb{N}} U_k^1 \times U_k^2 \in \Sigma$ *for* $U_k^i \in \Delta_i$*, by*

[2] $\theta(\alpha)$ is a measurable mapping between (M, Σ) and $\Delta(M, \Sigma)$. This is equivalent with the conditions on the two-variable *rate function* used in [13] to define continuous Markov processes (see, e.g. Proposition 2.9, of [10]).

$$\theta(a)((m_1,m_2))(S) = \sum_{(b,c)\in\mathcal{A}^2}^{b*c=a} \sum_{k\in K} \theta_1(b)(m_1)(U_k^1) \bullet \theta_2(c)(m_2)(U_k^2).$$

$\mathcal{K}$ represents the result of the synchronization of $\mathcal{K}_1$ and $\mathcal{K}_2$: θ calculates the rate of a by summing all the possible synchronizations $b * c = a$ between $\mathcal{K}_1$ and $\mathcal{K}_2$. The properties of $\bullet$ guarantee that the previous sum is convergent and independent of the choice of the bases. Because $\bullet$ is bilinear, $r \bullet 0 = 0$.

Lemma 1. *If $\mathcal{K}_1, \mathcal{K}_2 \in \mathfrak{K}$, then $\mathcal{K}_1 \times \mathcal{K}_2 \in \mathfrak{K}$.*

If $(\mathcal{K}_1, m_1)$ and $(\mathcal{K}_2, m_2)$ are CMPs, then $(\mathcal{K}_1 \times \mathcal{K}_2, (m_1, m_2))$ is a CMP called the *synchronization of* $(\mathcal{K}_1, m_1)$ and $(\mathcal{K}_2, m_2)$.

3.2 Parallel Composition

For introducing a concept of parallel composition that is general enough to include most of the similar concepts, we assume that the support set of the Markov kernel has an algebraic structure called *modular structure*.

Definition 4 (Modular structure). *A tuple $(M, \equiv, \otimes)$ is a* modular structure *on a set M if $\equiv \subseteq M \times M$ is an equivalence relation and $\otimes : M \times M \hookrightarrow M$ is a partial operation which, with respect to $\equiv$, is*

– *a* **congruence***: if $m_0 \equiv m_1$, then $m_0 \otimes m_2$ is defined iff $m_1 \otimes m_2$ is defined and*
$$m_0 \otimes m_2 \equiv m_1 \otimes m_2,$$
– **associative***: $(m_0 \otimes m_1) \otimes m_2$ is defined iff $m_0 \otimes (m_1 \otimes m_2)$ is defined and*
$$(m_0 \otimes m_1) \otimes m_2 \equiv m_0 \otimes (m_1 \otimes m_2),$$
– **commutative***: $m_0 \otimes m_1$ is defined iff $m_1 \otimes m_0$ is defined and*
$$m_0 \otimes m_1 \equiv m_1 \otimes m_0,$$
– **modular***: if $m_0 \otimes m_1 \equiv n_0 \otimes n_1$, then either $m_i \equiv n_j$ and $m_{1-i} \equiv n_{1-j}$ for $i, j \in \{0,1\}$, or there exist $m_j^i \in M$ for $i, j \in \{0,1\}$, such that $m_i \equiv m_0^i \otimes m_1^i$ and $n_i \equiv m_i^0 \otimes m_i^1$ for $i \in \{0,1\}$;*
– **wellfounded***: the relation $\{(m,n) \mid \exists n' \in M, m \equiv n \otimes n'\}$ is wellfounded.*

Process algebras are examples of modular structures where $\equiv$ is the structural congruence or some bisimulation relation, while $\otimes$ is, for instance, the parallel composition. In these cases well-foundedness expresses the fact that any process (modulo (Nil): $P \equiv P \otimes 0$) can be decomposed into a finite number of processes that cannot be, further, decomposed; and modularity guarantees the uniqueness of this decomposition up to structural congruence. In process algebras these hold, modulo (Nil), due to the inductive definition of the set of processes.

For modular structures, we lift the signature to sets by defining, for arbitrary $N, N' \subseteq M$, $N \otimes N' = \{m \in M \mid m \equiv n \otimes n' \text{ for some } n \in N, n' \in N'\}$. Moreover, if $\Sigma \subseteq 2^M$, let $\Sigma \otimes \Sigma = \{N \otimes N' \mid N, N' \in \Sigma\}$.

Definition 5 (Modular Markov process). *An $\mathcal{A}$-*modular Markov kernel *is a tuple $\mathcal{M} = (\mathcal{K}, \equiv, \otimes)$, where $\mathcal{K} = (M, \Sigma, \theta) \in \mathfrak{K}$ and $(M, \equiv, \otimes)$ is a modular structure such that its algebraic structure satisfy the following properties*

1. it preserves the Borel-algebras, i.e., $\Sigma \otimes \Sigma \subseteq \Sigma$,
2. it preserves the behaviours of modules and their synchronization, i.e.,
$(i).\ \equiv \subseteq \sim$, $(ii).\ (\mathcal{K}, m_0 \otimes m_1) \sim (\mathcal{K} \times \mathcal{K}, (m_0, m_1))$.
If $m \in M$, $(\mathcal{M}, m)$ *is a* modular Markov process.

Condition 2(ii) requires that $(\mathcal{K}, m_0 \otimes m_1)$ is bisimilar with the synchronization of $(\mathcal{K}, m_0)$ and $(\mathcal{K}, m_1)$.

M is called the *support of* $\mathcal{M}$, denoted $sup(\mathcal{M})$. Let $\mathfrak{M}$ be the class of $\mathcal{A}$-modular Markov kernels (MMKs); we use $\mathcal{M}, \mathcal{N}, \mathcal{M}_i, \mathcal{M}'$ to range over $\mathfrak{M}$.

Because MMKs preserves the synchronisation of the modules, stochastic bisimulation is a congruence.

Theorem 1 (Congruence). *Given* $(\mathcal{K}, \equiv, \otimes) \in \mathfrak{M}$, *if* $m \sim_{\mathcal{K}} m'$ *and both* $m \otimes n$ *and* $m' \otimes n$ *are defined, then* $m \otimes n \sim_{\mathcal{K}} m' \otimes n$.

4 Modular Markovian Logic

In this section we introduce Modular Markovian Logic (MML).

The formulas of MML are the elements of the set $\mathcal{L}$ introduced by the following grammar, for arbitrary $a \in \mathcal{A}$ and $r \in \mathbb{Q}_+$.

$$\phi := \top \ \vdots\ \neg\phi \ \vdots\ \phi \wedge \phi \ \vdots\ L^a_r\phi \ \vdots\ \phi|\phi.$$

The semantics is given by the *satisfiability relation* " $\Vdash$ " defined for $\mathcal{M} \in \mathfrak{M}$ and $m \in sup(\mathcal{M})$, inductively as follows.

$\mathcal{M}, m \Vdash \top$ always;
$\mathcal{M}, m \Vdash \neg\phi$ iff it is not the case that $\mathcal{M}, m \Vdash \phi$;
$\mathcal{M}, m \Vdash \phi \wedge \psi$ iff $\mathcal{M}, m \Vdash \phi$ and $\mathcal{M}, m \Vdash \psi$;
$\mathcal{M}, m \Vdash L^a_r\phi$ iff $\theta(a)(m)([\![\phi]\!]_{\mathcal{M}}) \geq r$, where $[\![\phi]\!]_{\mathcal{M}} = \{m \in M | \mathcal{M}, m \Vdash \phi\}$;
$\mathcal{M}, m \Vdash \phi_1|\phi_2$ iff $m \equiv m_1 \otimes m_2$ and $\mathcal{M}, m_i \Vdash \phi_i$, $1 = 1, 2$.

"|" is a polyadic modality of arity 2. The formula $L^a_r\phi$ is interpreted as *"the rate of an a-transition from the current state to a state satisfying* ϕ *is at least* r*"*. Notice that the semantics of $L^a_r\phi$ is well defined only if $[\![\phi]\!]_{\mathcal{M}}$ is measurable. This is guaranteed by the next lemma.

Lemma 2. *For any* $\phi \in \mathcal{L}$ *and any* $\mathcal{M} = (M, \Sigma, \theta) \in \mathfrak{M}$, $[\![\phi]\!]_{\mathcal{M}} \in \Sigma$.

When it is not the case that $\mathcal{M}, m \Vdash \phi$, we write $\mathcal{M}, m \nVdash \phi$. A formula ϕ is *satisfiable* if there exists $\mathcal{M} \in \mathfrak{M}$ and $m \in sup(\mathcal{M})$ such that $\mathcal{M}, m \Vdash \phi$. If $\neg\phi$ is not satisfiable, ϕ is *valid*, denoted by $\Vdash \phi$.

In what follows we consider all the Boolean derived operators. In addition, let $\left[\!\left|+\right|\!\right]_{i=1..n} \phi_i = \bigwedge_{\substack{i,j=1..n \\ i \neq j}} (\phi_i \to \neg\phi_j)$ and $\bot = \neg\top$ and for $k \in \mathbb{N}$, let $k = \neg(\underbrace{\top|\top|..|\top}_{k+1})$; notice that $\mathcal{M}, m \Vdash k$ iff m can be decomposed in maximum k modules.

In the rest of this section we focus on the logical equivalence induced by MML on MMPs and its relation to stochastic bisimulation on MMPs. The next theorem states that $\equiv$ preserves the satisfiability of $\mathcal{L}$ formulas.

Theorem 2. *For $\mathcal{M} \in \mathfrak{M}$ and $m, n \in \sup(\mathcal{M})$, if $m \equiv n$, then for all $\phi \in \mathcal{L}$, $\mathcal{M}, m \Vdash \phi$ iff $\mathcal{M}, n \Vdash \phi$.*

Let $\mathcal{L}^* \subsetneq \mathcal{L}$ be defined by the grammar $\phi := \top \vdots \neg\phi \vdots \phi \wedge \phi \vdots L^a_r\phi$. The next theorem reproduces a similar result presented in [13,28].

Theorem 3. *Given $\mathcal{M} \in \mathfrak{M}$ and $m, n \in \sup(\mathcal{M})$, if [for any $\phi \in \mathcal{L}^*$, $\mathcal{M}, m \Vdash \phi$ iff $\mathcal{M}, n \Vdash \phi$], then $m \sim n$.*

5 A Complete Hilbert-Style Axiomatization for MML

Tables 1, 2 and 3 contain a Hilbert-style axiomatization for MML.

The stochastic axioms in Table 1 have been proposed in [8] where we have proved that they form a complete axiomatization for CMPs. These axioms are similar, but more complex due to stochasticity, than the ones proposed in [31] for Harsanyi type spaces. As in the probabilistic case, we have infinitary rules (R2) and (R3) that encode the Archimedean properties of $\mathbb{Q}$. However, a finitary axiomatization is possible on the lines of [17] at the price of defining some complex operators, as shown in [22].

Table 1. Stochastic Axioms

(A1): $\vdash L^a_0\phi$
(A2): $\vdash L^a_{r+s}\phi \to L^a_r\phi$
(A3): $\vdash L^a_r(\phi \wedge \psi) \wedge L^a_s(\phi \wedge \neg\psi) \to L^a_{r+s}\phi$
(A4): $\vdash \neg L^a_r(\phi \wedge \psi) \wedge \neg L^a_s(\phi \wedge \neg\psi) \to \neg L^a_{r+s}\phi$
(R1): If $\vdash \phi \to \psi$ then $\vdash L^a_r\phi \to L^a_r\psi$
(R2): If $\forall r < s, \vdash \phi \to L^a_r\psi$ then $\vdash \phi \to L^a_s\psi$
(R3): If $\forall r > s, \vdash \phi \to L^a_r\psi$ then $\vdash \phi \to \bot$

Table 2. Structural Axioms

(A5): $\vdash (\phi|\psi)|\rho \to \phi|(\psi|\rho)$
(A6): $\vdash \phi|\psi \to \psi|\phi$
(A7): $\vdash \phi|\bot \to \bot$
(A8): $\vdash \phi|(\psi \vee \rho) \to (\phi|\psi \vee \phi|\rho)$
(R4): If $\vdash \phi \to \psi$ then $\vdash \phi|\rho \to \psi|\rho$
(R5): If $\vdash \phi \to \phi|\top$ then $\vdash \phi \to \bot$

The structural axioms in Table 2 are similar to the axioms proposed in [25] for a spatial logic on CCS semantics. The main difference is rule (R5) which rejects models that do not respect the modularity conditions. An example is the rule (Nil): $P \equiv P|0$ which allows processes with (trivial) non-wellfounded structure. However, one can easily make an MMP from a process algebra term by simply taking the quotient of the class of processes by (Nil) and similar rules.

To simplify the form of the modular axioms in Table 3, we use some additional notations. π_k is the set of permutations of $\{1, .., k\}$. For arbitrary $a \in \mathcal{A}$, we assume that the set of its $*$-decompositions (which is finite) is indexed and let $I_a = \{i \mid b_i * c_i = a\}$. If $\{(r^{i,j}_k) \mid i \in I, k \in K, j \in \{0,1\}\}, \{(s^{i,j}_k) \mid i \in I, k \in K, j \in \{0,1\}\} \subseteq \mathbb{Q}_+$, let $r^I_K \bullet s^I_K = \sum_{i \in I} \sum_{j \in \{0,1\}}^{k \in K} (r^{i,j}_k) \bullet (s^{i,j}_k)$.

The rules (R6) and (R7) encode the well-foundedness and the modularity of the models. The rules (R8) and (R9) are logical versions of classical *expansion laws* for parallel operator. (R8) states that the rate of the a-transitions from m to $[\![\rho]\!]$ is at least the sum, after $k \in K$, of all $\bullet$-products of the rates of b and c-transitions (for $a = b * c$)

from m_1 and m_2 to $[\![\phi_k^j]\!]$ and $[\![\phi_k^{1-j}]\!]$ respectively ($j = 0, 1$), given that $m \equiv m_1 \otimes m_2$ and $[\![\rho]\!]$ covers $\bigcup_{k\in K} [\![\phi_k^j]\!] \otimes [\![\phi_k^{1-j}]\!]$. For instance, $\vdash (L_r^b\phi \wedge L_u^c\psi)|(L_s^c\psi \wedge L_v^b\phi) \rightarrow L^{b*c}_{(r\bullet s)+(u\bullet v)}\phi|\psi$ and $\vdash L_r^b\top|L_s^c\top \rightarrow L^{b*c}_{r\bullet s}\top$ are instances of (R8). Similarly, (R9) states that the rate of the a-transitions from m to $[\![\rho]\!]$ is strictly bigger than the sum, after $k \in K$, of all $\bullet$-products of the rates of b and c-transitions (for $a = b * c$) from m_1 and m_2 to $[\![\phi_k^j]\!]$ and $[\![\phi_k^{1-j}]\!]$ respectively ($j = 0, 1$), given that $m \equiv m_1 \otimes m_2$ and $[\![\rho]\!]$ is covered by $\bigcup_{k\in K} [\![\phi_k^j]\!] \otimes [\![\phi_k^{1-j}]\!]$. $\vdash (\neg L_r^b\top \wedge \neg L_u^c\top)|(\neg L_s^c\top \wedge \neg L_v^b\top) \rightarrow \neg L^a_{(r\bullet s)+(u\bullet v)}(\top|\top)$ is an instance of (R9) given that b, c are the only actions such that $a = b * c$.

Table 3. Modular axioms

(R6): If I is finite and $\vdash 1 \rightarrow \bigvee_{i\in I} \phi_i$, then $\vdash k \rightarrow \bigvee_{i_j\in I}^{s<k} \phi_{i_1}|\ldots|\phi_{i_s}$.

(R7): If $\vdash \bigwedge_{i=1..k}^{j=0,1} (\phi_i^j \rightarrow 1)$ and $\vdash \phi_1^0|..|\phi_k^0 \rightarrow \phi_1^1|..|\phi_l^1$, then $k = l$ and

$$\vdash \bigvee_{\sigma\in\pi_k} \bigwedge_{i=1..k} \phi_i^0 \leftrightarrow \phi_{\sigma(i)}^1$$

(R8): If K is finite and $\vdash (\boxed{+}_{k\in K} \phi_k^0) \wedge (\boxed{+}_{k\in K} \phi_k^1) \wedge (\bigvee_{k\in K} \phi_k^0|\phi_k^1 \rightarrow \rho)$, then

$$\vdash \left(\bigwedge_{k\in K}^{i\in I_a} \bigwedge_{j=0,1} L^{b_i}_{(r_k^{i,j})}\phi_k^j\right)|\left(\bigwedge_{k\in K}^{i\in I_a} \bigwedge_{j=0,1} L^{c_i}_{(s_k^{i,j})}\phi_k^{1-j}\right) \rightarrow L^a_{(r_K^I \bullet s_K^I)}\rho$$

(R9): If K is finite and $\vdash (\boxed{+}_{k\in K} \phi_k^0) \wedge (\boxed{+}_{k\in K} \phi_k^1) \wedge (\rho \rightarrow \bigvee_{k\in K} \phi_k^0|\phi_k^1)$, then

$$\vdash \left(\bigwedge_{k\in K}^{i\in I_a} \bigwedge_{j=0,1} \neg L^{b_i}_{(r_k^{i,j})}\phi_k^j\right)|\left(\bigwedge_{k\in K}^{i\in I_a} \bigwedge_{j=0,1} \neg L^{c_i}_{(s_k^{i,j})}\phi_k^{1-j}\right) \rightarrow \neg L^a_{(r_K^I \bullet s_K^I)}\rho$$

As usual, we say that a formula ϕ is *provable*, denoted by $\vdash \phi$, if it can be proved from the axioms (using also Boolean rules). ϕ is *consistent*, if $\phi \rightarrow \bot$ is not provable. Given a set Φ, $\Psi \subseteq \mathcal{L}$, Φ proves Ψ if from the formulas of Φ and the axioms we can prove all $\psi \in \Psi$; we write $\Phi \vdash \Psi$. Φ is consistent if it is not the case that $\Phi \vdash \bot$. For a sublanguage $L \subseteq \mathcal{L}$, we call Φ *L-maximally consistent* if Φ is consistent and no formula of L can be added to it without making it inconsistent. For $\Lambda_1, \Lambda_2 \subseteq \mathcal{L}$, $\Lambda_1|\Lambda_2 = \{\phi_1|\phi_2 : \phi_i \in \Lambda_i, i = 1, 2\}$.

Theorem 4 (Soundness). *The axiomatic system of MML is sound for the Markovian semantics, i.e., for any $\phi \in \mathcal{L}$, if $\vdash \phi$ then $\Vdash \phi$.*

In what follows we prove the finite model property for MML by constructing a model for a given consistent formula. This result will eventually prove that the axiomatic system is also complete for the Markovian semantics, meaning that everything that is true for all the models can be proved. Before proceeding, we fix some notations.

For $n \in \mathbb{N}$, $n \neq 0$, let $\mathbb{Q}_n = \{\frac{p}{n} : p \in \mathbb{N}\}$. If $S \subseteq \mathbb{Q}$ is finite, the *granularity of* S, $gr(S)$, is the lowest common denominator of the elements of S.

The *modal depth of* $\phi \in \mathcal{L}$ is defined by $md(\top) = 0$, $md(\neg\phi) = md(\phi)$, $md(L^a_r\phi) = md(\phi) + 1$ and $md(\phi \wedge \psi) = md(\phi|\psi) = max(md(\phi), md(\psi))$.

The *structural depth of* $\phi \in \mathcal{L}$ is defined by $sd(\neg\phi) = sd(L^a_r\phi) = sd(\phi)$, $sd(\phi \wedge \psi) = max(sd(\phi), sd(\psi))$ and $sd(\phi|\psi) = sd(\phi) + sd(\psi) + 1$.

The *granularity of* $\phi \in \mathcal{L}$ is $gr(\phi) = gr(R)$, where $R \subseteq \mathbb{Q}_+$ is the set of indexes r of the operators L^a_r present in ϕ; the *upper bound of* ϕ is $max(\phi) = max(R)$.

For arbitrary $n \in \mathbb{N}$, let $\mathcal{L}_n$ be the sublanguage of $\mathcal{L}$ that uses only modal operators L^a_r with $r \in \mathbb{Q}_n$. For $\Lambda \subseteq \mathcal{L}$, let $[\Lambda]_n = \Lambda \cup \{\phi \in \mathcal{L}_n : \Lambda \vdash \phi\}$.

Consider a consistent formula $\psi \in \mathcal{L}$ with $gr(\psi) = n$ and $sd(\psi) = e$.
Let $\mathcal{L}[\psi] = \{\phi \in \mathcal{L}_n \mid max(\phi) \leq max(\psi), md(\phi) \leq md(\psi)$ and $sd(\phi) \leq sd(\psi)\}$.

In what follows we construct $\mathcal{M}_\psi \in \mathfrak{M}$ such that each $\Gamma \in \sup(\mathcal{M}_\psi)$ is a consistent set of formulas that contains an $\mathcal{L}[\psi]$-maximally consistent set and each $\mathcal{L}[\psi]$-maximally consistent set is contained in some $\Gamma \in \sup(\mathcal{M}_\psi)$. And we prove the truth lemma stating that for any $\phi \in \mathcal{L}[\psi]$, $\phi \in \Gamma$ iff $\mathcal{M}_\psi, \Gamma \Vdash \phi$.

Let $\Omega[\psi]$ be the set of $\mathcal{L}[\psi]$-maximally consistent sets of formulas. $\Omega[\psi]$ is finite and any $\Lambda \in \Omega[\psi]$ contains finitely many nontrivial formulas[3]; in the rest of this construction we only count non-trivial formulas while ignoring the rest.

For each $\Lambda \in \Omega[\psi]$, such that $\{\phi_1, ..., \phi_i\}$ is the set of its non-trivial formulas, we construct $\Lambda^+ \supseteq [\Lambda]_n$ with the property that $\forall\phi \in \Lambda$ and $a \in \mathcal{A}$ there exists $\neg L^a_r\phi \in \Lambda^+$.

The step [ϕ_1 and Λ:] (R3) guarantees that $\exists r \in \mathbb{Q}_n$ s.t. $[\Lambda]_n \cup \{\neg L^a_r\phi_1\}$ is consistent. Let $y^a_1 = min\{s \in \mathbb{Q}_n : [\Lambda]_n \cup \{\neg L^a_s\phi_1\}$ is consistent$\}$ and $x^a_1 = max\{s \in \mathbb{Q}_n : L^a_s\phi_1 \in [\Lambda]_n\}$ ((R3) guarantees the existence of max). (R2) implies that $\exists r \in \mathbb{Q} \setminus \mathbb{Q}_n$ s.t., $x^a_1 < r < y^a_1$ and $\{\neg L^a_r\phi_1\} \cup [\Lambda]_n$ is consistent. Let $n_1 = gran\{1/n, r\}$. Let $s^a_1 = min\{s \in \mathbb{Q}_{n_1} : [\Lambda]_{n_1} \cup \{\neg L^a_s\phi_1\}$ is consistent$\}$, $\Lambda^a_1 = \Lambda \cup \{\neg L^a_{s_1}\phi_1\}$ and $\Lambda_1 = \bigcup_{a \in A} \Lambda^a_1$.

We repeat this step of the construction for $[\phi_2$ and $\Lambda_1]$,..,$[\phi_i$ and $\Lambda_{i-1}]$ and we obtain $\Lambda \subseteq \Lambda_1 \subseteq ... \subseteq \Lambda_i$, where Λ_i is a consistent set containing a finite set of nontrivial formulas. Let $n_\Lambda = gran\{1/n_1, .., 1/n_i\}$. We make this construction for all $\Lambda \in \Omega[\psi]$. Let $\nu = gran\{1/n_\Lambda : \Lambda \in \Omega[\psi]\}$. Let $\Lambda^+ = [\Lambda_i]_\nu$ and $\Omega^+[\psi] = \{\Lambda^+ : \Lambda \in \Omega[\psi]\}$. Notice that $\nu > n$; we call ν the *parameter of* $\Omega[\psi]$.

Remark 1. For each $\Lambda \in \Omega[\psi]$, $\phi \in \Lambda$ and $a \in \mathcal{A}$, there exist $s, t \in \mathbb{Q}_\nu$, $s < t$, such that $L^a_s\phi, \neg L^a_r\phi \in \Gamma^+$. Moreover, there exists $f \in \Lambda^+$ such that $f \vdash \Lambda^+$.

Let Ω_ν be the set of $\mathcal{L}_\nu$-maximally consistent sets of formulas and $\sigma : \Omega^+[\psi] \to \Omega_\nu$ be an injection such that for any $\Lambda^+ \in \Omega^+[\psi]$, $\Lambda^+ \subseteq \sigma(\Lambda^+)$. Let $\Omega_\nu[\psi] = \sigma(\Omega^+[\psi])$, and for $\phi \in \mathcal{L}[\psi]$, let $[\![\phi]\!] = \{\Gamma \in \Omega_\nu[\psi] : \phi \in \Gamma\}$. For an arbitrary $\Gamma \in \Omega_\nu[\psi]$, if $\Gamma = \sigma(\Lambda^+)$ for $\Lambda \in \Omega[\psi]$, we denote Λ by $\overline{\Gamma}$.

Lemma 3. *(1)* $\Omega_\nu[\psi]$ *is finite.* *(2)* $2^{\Omega_\nu[\psi]} = \{[\![\phi]\!] \mid \phi \in \mathcal{L}[\psi]\}$.
(3) For any $\phi_1, \phi_2 \in \mathcal{L}[\psi]$, $\vdash \phi_1 \to \phi_2$ *iff* $[\![\phi_1]\!] \subseteq [\![\phi_2]\!]$.
(4) For any $\Gamma \in \Omega_\nu[\psi]$, $\phi \in \mathcal{L}[\psi]$ *and* $a \in \mathcal{A}$, *there exist*
$x = max\{r \in \mathbb{Q}_\nu : L^a_r\phi \in \Gamma\}$, $y = min\{r \in \mathbb{Q}_\nu : \neg L^a_r\phi \in \Gamma\}$ *and* $y = x + 1/\nu$.

[3] By nontrivial formulas we mean the formulas that are not obtained from more basic consistent ones by boolean derivations.

Let Ω be the set of $\mathcal{L}$-maximally consistent sets of formulas and $\pi : \Omega_v \to \Omega$ an injection such that for any $\Gamma \in \Omega_v$, $\Gamma \subseteq \pi(\Gamma)$.

Lemma 4. *For any $\Gamma \in \Omega_v[\psi]$, any $\phi \in \mathcal{L}[\psi]$ and any $a \in \mathcal{A}$, there exist $x^\infty = sup\{r \in \mathbb{Q} : L^a_r\phi \in \pi(\Gamma)\} = inf\{r \in \mathbb{Q} : \neg L^a_r\phi \in \pi(\Gamma)\}$ and $x \leq x^\infty < y$.*

We denote by $a^\Gamma_\phi = x^\infty$ defined for $\phi \in \mathcal{L}[\psi]$, $\Gamma \in \Omega_v[\psi]$ and $a \in \mathcal{A}$.

Lemma 5. *$\mathcal{M}_\psi = (\mathcal{K}_\psi, =, \otimes) \in \mathfrak{M}$, where $\mathcal{K}_\psi = (\Omega_v[\psi], 2^{\Omega_v[\psi]}, \theta_\psi)$ with*
(i). θ_ψ defined for arbitrary $\phi \in \mathcal{L}[\psi]$, $\Gamma \in \Omega_v[\psi]$, $a \in \mathcal{A}$ by $\theta_\psi(a)(\Gamma)([\![\phi]\!]) = a^\Gamma_\phi$,
(ii). $\otimes$ defined for $\Gamma, \Gamma', \Gamma'' \in \Omega_v[\psi]$ by $[\Gamma = \Gamma' \otimes \Gamma''$ iff $\overline{\Gamma'}|\overline{\Gamma''} \subseteq \overline{\Gamma}]$.

Proof. This proof is rather complex. we only sketch here the main arguments.
Lemma 3(3) proves that for arbitrary $\Gamma \in \Omega_v[\psi]$ and $a \in \mathcal{A}$, $\theta_\psi(a)(\Gamma)$ is well defined.

To prove that $\theta_\psi(a)(\Gamma)$ is a measure, we show that $\theta_\psi(a)(\Gamma)([\![\bot]\!]) = 0$ and that for $\phi_1, \phi_2 \in \mathcal{L}[\psi]$ with $\vdash \phi_1 \to \neg\phi_2$, $\theta_\psi(a)(\Gamma)([\![\phi_1]\!]) + \theta_\psi(a)(\Gamma)([\![\phi_2]\!]) = \theta_\psi(a)(\Gamma)([\![\phi_1 \vee \phi_2]\!])$. These use the stochastic and the structural axioms, especially the archimedian rules.

The modular structure of $\mathcal{M}_\psi$ is proved based on Rules (R5), (R6) and (R7).

It remains to prove that $(\mathcal{M}_\psi, \Gamma' \otimes \Gamma'') \sim (\mathcal{M}_\psi \times \mathcal{M}_\psi, (\Gamma', \Gamma''))$. This requires to prove that $\overline{\Gamma'}|\overline{\Gamma''} \subseteq \overline{\Gamma}$ implies that for arbitrary $\phi \in \mathcal{L}[\psi]$,

$$\theta_\psi(a)(\Gamma)([\![\phi]\!]) = \sum_{b * c = a} \sum_{[\![\phi]\!] = [\![g'|g'']\!]}^{g', g'' \in \mathcal{F}^*} \theta_\psi(b)(\Gamma')([\![g']\!]) \bullet \theta_\psi(c)(\Gamma'')([\![g'']\!]).$$

This prove is done by involving the Rules (R8) and (R9) that approximates, from below and from above the value of $\theta_\psi(a)(\Gamma)([\![\phi]\!])$. Also here the archimedian rules play a central role together with the hypothesis of the continuity of $\bullet$.

Now we can prove the Truth Lemma.

Lemma 6 (Truth Lemma). *If $\phi \in \mathcal{L}[\psi]$ and $\Gamma \in \Omega_v[\psi]$, then $[\mathcal{M}_\psi, \Gamma \Vdash \phi$ iff $\phi \in \Gamma]$.*

Proof. Induction on the structure of ϕ. The Boolean cases are trivial.
The case $\phi = L^a_r\phi'$**:** ($\Longrightarrow$) Suppose that $\mathcal{M}_\psi, \Gamma \Vdash L^a_r\phi'$ and $L^a_r\phi' \notin \Gamma$. Because Γ is $\mathcal{L}[\psi]$-maximally consistent, $\neg L^a_r\phi' \in \Gamma$. Let $y = min\{r \in \mathbb{Q}_p : \neg L^a_r\phi' \in \Gamma\}$. Then, from $\neg L^a_r\phi' \in \Gamma$, we obtain $r \geq y$. But $\mathcal{M}_\psi, \Gamma \Vdash L^a_r\phi'$ is equivalent with $\theta_\psi(a)(\Gamma)([\![\phi']\!]) \geq r$, i.e. $a^\Gamma_{\phi'} \geq r$. On the other hand, in Lemma 4 we proved that $a^\Gamma_{\phi'} < y$ - contradiction.
($\Longleftarrow$) Suppose that $L^a_r\phi' \in \Gamma$. Then $r \leq a^\Gamma_\phi$, implying $\theta_\psi(a)(\Gamma)([\![\phi]\!]) \geq r$.
The case $\phi = \phi_1|\phi_2$**:** ($\Longrightarrow$) If $\mathcal{M}_\psi, \Gamma \Vdash \phi_1|\phi_2$, then $\Gamma = \Gamma_1 \otimes \Gamma_2$ and $\mathcal{M}_\psi, \Gamma_i \Vdash \phi_i$, $i = 1, 2$. The inductive hypothesis implies that $\phi_i \in \Gamma_i$ and because $\Gamma_1|\Gamma_2 \subseteq \Gamma$, $\phi_1|\phi_2 \in \Gamma$.
($\Longleftarrow$) If $\phi_1|\phi_2 \in \Gamma$, then there exist Γ_i with $\phi_i \in \Gamma_i$ and $\Gamma_1|\Gamma_2 \subseteq \Gamma$, i.e. $\Gamma = \Gamma_1 \otimes \Gamma_2$.

The previous lemma implies the small model property for our logic.

Theorem 5 (Small model property). *For any consistent formula ϕ, there exists $\mathcal{M} \in \mathfrak{M}$ with the cardinality of $sup(\mathcal{M})$ bound by the structure of ϕ, and $m \in sup(\mathcal{M})$ such that $\mathcal{M}, m \Vdash \phi$.*

The small model property proves the completeness of the axiomatic system.

Theorem 6 (Completeness). *MML is complete with respect to the Markovian semantics, i.e. if* $\Vdash \psi$*, then* $\vdash \psi$*.*

Proof. The proof is based on the fact that any consistent formula has a model. Indeed, [$\Vdash \psi$ implies $\vdash \psi$] is equivalent with [$\nvdash \psi$ implies $\nVdash \psi$], that is equivalent with [the consistency of $\neg\psi$ implies that there exists a model $\mathcal{M}$ such that $\mathcal{M}, m \nVdash \psi$], that is equivalent with [the consistency of $\neg\psi$ implies the satisfiability of $\neg\psi$].

6 Conclusions and Future Work

In this paper we have introduced Modular Markovian Logic, a new logic that combines features of stochastic and modular logics. Its semantics is in terms of modular Markov processes which are compositional continuous-time and continuous-space Markov processes. MML is appropriate for specifying and verifying modular properties of stochastic systems and to prove global properties from local properties of subsystems. For instance modular proof rules as the ones below can be given as instances of (R9).

$$\frac{P \Vdash L^b_r \top, \quad P' \Vdash L^c_s \top}{P \otimes P' \Vdash L^{b*c}_{r \bullet s} \top} \quad \text{and} \quad \frac{P \Vdash L^b_r \phi \wedge L^c_u \psi, \quad P' \Vdash L^c_s \psi \wedge L^b_v \phi}{P \otimes P' \Vdash L^{b*c}_{(r \bullet s)+(u \bullet v)} \rho} \, (\vdash \phi | \psi \to \rho).$$

Similarly, if b, c are unique such that $a = b * c$ and P, P' are unique such that $P'' \equiv P \otimes P'$, the rule below is based on an instance of (R10).

$$\frac{P \Vdash \neg L^b_r \top \wedge \neg L^c_u \top, \quad P' \Vdash \neg L^c_s \top \wedge \neg L^b_v \top}{P'' \Vdash \neg L^a_{(r \bullet s)+(u \bullet v)} (\top | \top)}$$

In this paper we have presented a complete Hilbert-style axiomatization for MML and prove the small model property. For future work we intend to focus on decidability and complexity problems following the line of [24], as well as on axiomatizations of model checking and possible procedures to automatize the proof of modular rules.

References

1. Aumann, R.: Interactive epistemology: Probability. Int. J. Game Theory 28 (1999)
2. Bergstra, J.A., Klop, J.W.: ACP: A Universal Axiom System for Process Specification. CWI Quarterly 15 (1987)
3. Blute, R., Desharnais, J., Edalat, A., Panangaden, P.: Bisimulation for labeled Markov processes. In: Proc. of LICS 1997. IEEE Press, Los Alamitos (1997)
4. Bravetti, M., Hermanns, H., Katoen, J.-P.: YMCA: Why Markov Chain Algebra? ENTCS 162 (2006)
5. Caires, L., Cardelli, L.: A Spatial Logic for Concurrency. Inf. and Comp. 186(2) (2003)
6. Cardelli, L., Gordon, A.: Anytime, anywhere: Modal logics for mobile ambients. In: Proc. of POPL 2000 (2000)
7. Cardelli, L., Mardare, R.: The Measurable Space of Stochastic Processes. In: QEST 2010. IEEE Press, Los Alamitos (2010)
8. Cardelli, L., Mardare, R.: Continuous Markovian Logic. Tech.Rep. TR-4-2010, The Microsoft Research-CoSBi Centre (2010)

9. Cirstea, C., Pattinson, D.: Modular proof systems for coalgebraic logics. TCS 388 (2007)
10. Doberkat, E.: Stochastic Relations. Foundations for Markov Transition Systems. Chapman and Hall/CRC (2007)
11. Desharnais, J., Edalat, A., Panangaden, P.: A logical characterization of bisimulation for labeled Markov processes. In: Proc of LICS 1998. IEEE Press, Los Alamitos (1998)
12. Desharnais, J., Edalat, A., Panangaden, P.: Bisimulation for labeled Markov Processes. Inf. and Comp. 179(2) (2002)
13. Desharnais, J., Panangaden, P.: Continous Stochastic Logic Characterizes Bisimulation of Continuous-time Markov Processes. JLAP 56 (2003)
14. Fagin, R., Halpern, J.: Reasoning about knowledge and probability. J. ACM 41 (1994)
15. Harsanyi, J.: Games with incomplete information played by bayesian players, part one. Management Sci. 14 (1967)
16. Hillston, J.: Process algebras for quantitative analysis. In: Proc. LICS 2005. IEEE Press, Los Alamitos (2005)
17. Heifetz, A., Mongin, P.: Probability Logic for Type Spaces. Games and Economic Behavior 35 (2001)
18. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. JACM 32 (1985)
19. Hoare, C.A.R.: Communicating sequential processes. Communications of the ACM 21(8) (1978)
20. Httel, H., Larsen, K.G.: The Use of Static Constructs in A Modal Process Logic. Logic at Botik (1989)
21. Kozen, D.: Results on the Propositional Mu-Calculus. TCS 27(3) (1983)
22. Kupke, C., Pattinson, D.: On Modal Logics of Linear Inequalities. In: Proc of AiML 2010 (2010)
23. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Inf. and Comp. 94 (1991)
24. Mardare, R.: Decidability of Modular Logics for Concurrency. In: Proc. of PS 2011 (2011)
25. Mardare, R., Policriti, A.: A Complete Axiomatic System for a Process-Based Spatial Logic. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 491–502. Springer, Heidelberg (2008)
26. Milner, R.: A calculus of communicating systems. Journal of the ACM (1980)
27. Moss, L.S., Viglizzo, I.D.: Harsanyi type spaces and final coalgebras constructed from satisfied theories. ENTCS 106 (2004)
28. Panangaden, P.: Labelled Markov Processes. Imperial College Press, London (2009)
29. Priami, C.: Stochastic π-Calculus. Computer Journal 38(7) (1995)
30. Reynolds, J.C.: Separation Logic: A Logic for Shared Mutable Data Structures. In: Proc. of LICS 2002. IEEE Press, Los Alamitos (2002)
31. Zhou, C.: A complete deductive system for probability logic with application to Harsanyi type spaces, PhD thesis, Indiana University (2007)

Programming with Infinitesimals: A WHILE-Language for Hybrid System Modeling*

Kohei Suenaga[1] and Ichiro Hasuo[2]

[1] JSPS Research Fellow, Kyoto University, Japan
[2] University of Tokyo, Japan

Abstract. We add, to the common combination of a WHILE-language and a Hoare-style program logic, a constant `dt` that represents an infinitesimal (i.e. infinitely small) value. The outcome is a framework for modeling and verification of *hybrid systems*: hybrid systems exhibit both continuous and discrete dynamics and getting them right is a pressing challenge. We rigorously define the semantics of programs in the language of *nonstandard analysis*, on the basis of which the program logic is shown to be sound and relatively complete.

1 Introduction

Hybrid systems are systems that deal with both discrete and continuous data. They have rapidly gained importance since more and more physical systems—cars, airplanes, etc.—are controlled with computers. Their sensors will provide physical, continuous data, while the behavior of controller software is governed by discrete data. Those information systems which interact with a physical ambience are more generally called *cyber-physical systems (CPS)*; hybrid systems are an important building block of CPSs.

Towards the goal of getting hybrid systems right, the research efforts have been mainly from two directions. *Control theory*—originally focusing on continuous data and their control e.g. via integration and differentiation—is currently extending its realm towards hybrid systems. The de facto standard SIMULINK tool for hybrid system modeling arises from this direction; it employs the block diagram formalism and offers simulation functionality—aiming at *testing* rather than *verification*. The current work is one of the attempts from the other direction—from the *formal verification* community—that advance from discrete to continuous.

Hybrid systems exhibit two kinds of dynamics: continuous *flow* and discrete *jump*. Hence for a formal verification approach to hybrid systems, the challenge is: 1) to incorporate flow-dynamics; and 2) to do so at the lowest possible cost, so that the discrete framework smoothly transfers to hybrid situations. A large body of existing work includes differential equations explicitly in the syntax; see the discussion of related work below. What we propose, instead, is to introduce a constant `dt` for an *infinitesimal* (i.e. infinitely small) value and *turn flow into jump*. With `dt`, the continuous operation of integration can be represented by a `while` loop, to which existing discrete techniques like Hoare-style program logics readily apply. For a rigorous mathematical development we employ *nonstandard analysis (NSA)* beautifully formalized by Robinson.

* We are greteful to Naoki Kobayashi and Toshimitsu Ushio for helpful discussions.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 392–403, 2011.

Concretely, in this paper we take the common combination of a WHILE-language, a first-order assertion language and a Hoare logic (e.g. in the textbook [10]); and add a constant dt to obtain a modeling and verification framework for hybrid systems. Its three ingredients are called WHILE$^{\mathtt{dt}}$, ASSN$^{\mathtt{dt}}$ and HOARE$^{\mathtt{dt}}$. These are connected by denotational semantics defined in the language of NSA. We prove soundness and relative completeness of the logic HOARE$^{\mathtt{dt}}$. Underlying the technical development is the idea of what we call *sectionwise execution*, illustrated by the following example.

Example 1.1 Let c_{elapse} be the following program; $\equiv$ denotes the syntactic equality.

$$c_{\mathsf{elapse}} \quad :\equiv \quad \left[\quad t := 0 \,; \quad \mathtt{while}\ t \leq 1\ \mathtt{do}\ t := t + \mathtt{dt} \quad \right]$$

The value designated by dt is infinitesimal; therefore the while loop will not terminate within finitely many steps. Nevertheless it is intuitive to expect that after an "execution" of this program (which takes an infinitely long time), the value of t should be infinitely close to 1. How can we turn this intuition into a mathematical argument?

Our idea is to think about *sectionwise execution*. For each natural number i we consider the *i-th section* of the program c_{elapse}, denoted by $c_{\mathsf{elapse}}|_i$. Concretely, $c_{\mathsf{elapse}}|_i$ is defined by replacing the infinitesimal dt in c_{elapse} by $\frac{1}{i+1}$:

$$c_{\mathsf{elapse}}|_i \quad :\equiv \quad \left[\quad t := 0 \,; \quad \mathtt{while}\ t \leq 1\ \mathtt{do}\ t := t + \tfrac{1}{i+1} \quad \right] .$$

Informally $c_{\mathsf{elapse}}|_i$ is the "i-th approximation" of the original c_{elapse}.

A section $c_{\mathsf{elapse}}|_i$ does terminate within finite steps and yields $1 + \frac{1}{i+1}$ as the value of t. Now we collect the outcomes of sectionwise executions and obtain a sequence

$$(1+1,\ 1+\tfrac{1}{2},\ 1+\tfrac{1}{3},\ \ldots,\ 1+\tfrac{1}{i},\ \ldots)$$

which is thought of as an incremental approximation of the actual outcome of the original program c_{elapse}. Indeed, in the language of NSA, the sequence represents a *hyperreal number* $\boldsymbol{r}$ that is infinitely close to 1.

We note that, as is clear from this example, a program of WHILE$^{\mathtt{dt}}$ is not executable in general. We would rather regard WHILE$^{\mathtt{dt}}$ as a modeling language for hybrid systems, with a merit of being close to a common programming style.

The idea of *turning flow into jump* with dt and NSA seems applicable to other discrete modeling/verification techniques than while-language and Hoare logic. We wish to further explore this potentiality. Adaptation of more advanced techniques for deductive program verification—such as invariant generation and type systems—to the presence of dt is another important direction of future work.

Due to the lack of space all the proofs are deferred to the extended version [9].

Related work There have been extensive research efforts towards hybrid systems from the formal verification community. Unlike the current work where we turn flow into jump via dt, most of them feature acute distinction between flow- and jump-dynamics.

Hybrid automaton [1] is one of the most successful approaches to verification of hybrid systems. A number of model-checking algorithms have been invented for automatic verification. The deductive approach in the current work—via theorem-proving in

HOARE$^{\mathtt{dt}}$—has an advantage of handling parameters (i.e. universal quantifiers or free variables that range over an infinite domain) well; model checking in such a situation necessarily calls for some abstraction technique. Our logic HOARE$^{\mathtt{dt}}$ is compositional too: a property of a whole system is inferred from the ones of its constituent parts.

Deductive verification of hybrid systems has seen great advancement through a recent series of work by Platzer and his colleagues, including [5, 6]. Their formalism is variations of *dynamic logic*, augmented with differential equations to incorporate flow-dynamics. They have also developed advanced techniques aimed at automated theorem proving, resulting in a sophisticated tool called KeYmaera [7]. We expect our approach (namely incorporating flow-dynamics via `dt`) offer a smoother transfer of existing discrete verification techniques to hybrid applications. Additionally, our preliminary observations suggest that some of the techniques developed by Platzer and others can be translated into the techniques that work in our framework.

The use of NSA as a foundation of hybrid system modeling is not proposed for the first time; see e.g. [8, 2]. Compared to these existing work, we claim our novelty is a clean integration of NSA and the widely-accepted programming style of `while`-languages, with an accompanying verification framework by HOARE$^{\mathtt{dt}}$.

2 A Nonstandard Analysis Primer

This section is mainly for fixing notations. For more details see e.g. [4].

The type of statements "there exists $i_0 \in \mathbb{N}$ such that, for each $i \geq i_0$, $\varphi(i)$ holds" is typical in analysis. It is often put as "for sufficiently large $i \in \mathbb{N}$." This means: the set $\{i \in \mathbb{N} \mid \varphi(i) \text{ holds}\} \subseteq \mathbb{N}$ belongs to the family $\mathcal{F}_0 := \{S \subseteq \mathbb{N} \mid \mathbb{N} \setminus S \text{ is finite}\}$.

In NSA, the family $\mathcal{F}_0$ is extended to so-called an *ultrafilter* $\mathcal{F}$. The latter is a convenient domain of "i-indexed truth values": notably for each set $S \subseteq \mathbb{N}$, exactly one out of S and $\mathbb{N} \setminus S$ belongs to $\mathcal{F}$.

Definition 2.1 (Ultrafilter) A *filter* is a family $\mathcal{F} \subseteq \mathcal{P}(\mathbb{N})$ such that: 1) $X \in \mathcal{F}$ and $X \subseteq U$ implies $U \in \mathcal{F}$; 2) $X \cap Y \in \mathcal{F}$ if $X, Y \in \mathcal{F}$. A nonempty filter $\mathcal{F} \neq \emptyset$ is said to be *proper* if it does not contain $\emptyset \subseteq \mathbb{N}$; equivalently, if $\mathcal{F} \neq \mathcal{P}(\mathbb{N})$. An *ultrafilter* is a maximal proper filter; equivalently, it is a filter $\mathcal{F}$ such that for each $S \subseteq \mathbb{N}$, exactly one out of S and $\mathbb{N} \setminus S$ belongs to $\mathcal{F}$.

A filter $\mathcal{F}'$ can be always extended to an ultrafilter $\mathcal{F} \supseteq \mathcal{F}'$; this is proved using Zorn's lemma. Since the family $\mathcal{F}_0$ is easily seen to be a filter, we have:

Lemma 2.2 *There is an ultrafilter $\mathcal{F}$ that contains $\mathcal{F}_0 = \{S \subseteq \mathbb{N} \mid \mathbb{N} \setminus S \text{ is finite}\}$.*

Throughout the rest of the paper we fix such $\mathcal{F}$. Its properties to be noted: 1) $\mathcal{F}$ is closed under finite intersections and infinite unions; 2) exactly one of S or $\mathbb{N} \setminus S$ belongs to $\mathcal{F}$, for each $S \subseteq \mathbb{N}$; and 3) if S is such that $\mathbb{N} \setminus S$ is finite, then $S \in \mathcal{F}$.

We say "$\varphi(i)$ holds for almost every $i \in \mathbb{N}$" for the fact that the set $\{i \mid \varphi(i) \text{ holds}\}$ belongs to $\mathcal{F}$. For its negation we say "for negligibly many i."

Definition 2.3 (Hypernumber $\boldsymbol{d} \in {}^*\mathbb{D}$) For a set $\mathbb{D}$ (typically it is $\mathbb{N}$ or $\mathbb{R}$), we define the set ${}^*\mathbb{D}$ by ${}^*\mathbb{D} := \mathbb{D}^{\mathbb{N}}/\sim_{\mathcal{F}}$. It is the set of infinite sequences on $\mathbb{D}$ modulo the following equivalence $\sim_{\mathcal{F}}$: we define $(d_0, d_1, \dots) \sim_{\mathcal{F}} (d'_0, d'_1, \dots)$ by

$$d_i = d'_i \text{ “for almost every } i\text{,” that is, } \{i \in \mathbb{N} \mid d_i = d'_i\} \in \mathcal{F}.$$

An equivalence class $\big[(d_i)_{i\in\mathbb{N}}\big]_{\sim_{\mathcal{F}}} \in {}^*\mathbb{D}$ shall be also denoted by $\big[(d_i)_{i\in\mathbb{N}}\big]$ or $(d_i)_{i\in\mathbb{N}}$ when no confusion occurs. An element $\boldsymbol{d} \in {}^*\mathbb{D}$ is called a *hypernumber*; in contrast $d \in \mathbb{D}$ is a *standard number*. Hypernumbers will be denoted in boldface like $\boldsymbol{d}$.

We say that $(d_i)_{i\in\mathbb{N}}$ is a *sequence representation* of $\boldsymbol{d} \in {}^*\mathbb{D}$ if $\boldsymbol{d} = [(d_i)_i]$. Note that, given $\boldsymbol{d} \in {}^*\mathbb{D}$, its sequence representation is not unique. There is a canonical embedding $\mathbb{D} \hookrightarrow {}^*\mathbb{D}$ mapping d to $[(d, d, \dots)]$; the latter shall also be denoted by d.

Definition 2.4 (Operations and relations on ${}^*\mathbb{D}$) An operation $f : \mathbb{D}^k \to \mathbb{D}$ of any finite arity k (such as $+ : \mathbb{R}^2 \to \mathbb{R}$) has a canonical “pointwise” extension $f : ({}^*\mathbb{D})^k \to {}^*\mathbb{D}$. A binary relation $R \subseteq \mathbb{D}^2$ (such as $<$ on real numbers) also extends to $R \subseteq ({}^*\mathbb{D})^2$.

$$f\big(\big[(d_i^{(0)})_{i\in\mathbb{N}}\big], \dots, \big[(d_i^{(k-1)})_{i\in\mathbb{N}}\big]\big) := \big[\big(f(d_i^{(0)}, \dots, d_i^{(k-1)})\big)_{i\in\mathbb{N}}\big],$$
$$\big[(d_i)_{i\in\mathbb{N}}\big] \; R \; \big[(d'_i)_{i\in\mathbb{N}}\big] \overset{\text{def.}}{\iff} d_i \; R \; d'_i \quad \text{for almost every } i.$$

These extensions are well-defined since $\mathcal{F}$ is closed under finite intersections.

Example 2.5 (ω and ω^{-1}) By ω we denote the hypernumber $\omega = [(1, 2, 3, \dots)] \in {}^*\mathbb{N}$. It is bigger than (the embedding of) any (standard) natural number $n = [(n, n, \dots)]$, since we have $n < i$ for all i except for finitely many. The presence of ω shows that $\mathbb{N} \subsetneq {}^*\mathbb{N}$ and $\mathbb{R} \subsetneq {}^*\mathbb{R}$. Its inverse $\omega^{-1} = [(1, \frac{1}{2}, \frac{1}{3}, \dots)]$ is positive ($0 < \omega^{-1}$) but is smaller than any (standard) positive real number $r > 0$.

These hypernumbers—*infinite* ω and *infinitesimal* ω^{-1}—will be heavily used.

For the set $\mathbb{B} = \{\mathrm{tt}, \mathrm{ff}\}$ of Boolean truth values we have the following. Therefore a “hyper Boolean value” does not make sense.

Lemma 2.6 *Assume that $\mathbb{D}$ is a finite set $\mathbb{D} = \{a_1, \dots, a_n\}$. Then the canonical inclusion map $\mathbb{D} \hookrightarrow {}^*\mathbb{D}$ is bijective. In particular we have ${}^*\mathbb{B} \cong \mathbb{B}$ for $\mathbb{B} = \{\mathrm{tt}, \mathrm{ff}\}$.* □

3 Programming Language WHILE$^{\mathtt{dt}}$

3.1 Syntax

We fix a countable set $\mathbf{Var}$ of *variables*.

Definition 3.1 (WHILE$^{\mathtt{dt}}$, WHILE) The syntax of our target language WHILE$^{\mathtt{dt}}$ is:

$$\begin{aligned}
\mathbf{AExp} \ni \quad & a ::= x \mid \mathtt{c}_r \mid a_1 \mathtt{\ aop\ } a_2 \mid \mathtt{dt} \mid \infty \\
& \text{where } x \in \mathbf{Var}, \mathtt{c}_r \text{ is a constant for } r \in \mathbb{R}, \text{ and } \mathtt{aop} \in \{+, -, \cdot, \hat{}\} \\
\mathbf{BExp} \ni \quad & b ::= \mathtt{true} \mid \mathtt{false} \mid b_1 \wedge b_2 \mid \neg b \mid a_1 < a_2 \\
\mathbf{Cmd} \ni \quad & c ::= \mathtt{skip} \mid x := a \mid c_1; c_2 \mid \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 \mid \mathtt{while}\ b\ \mathtt{do}\ c
\end{aligned}$$

An expression in **AExp** is said to be *arithmetic*; one in **BExp** is *Boolean* and one in **Cmd** is a *command*. The operator $a^{\wedge}b$ designates "a to the power of b" and will be denoted by a^b. The operator $^{\wedge}$ is included as a primitive for the purpose of relative completeness (Thm. 5.4). We will often denote the constant $\mathtt{c}_r$ by r.

By WHILE, we denote the fragment of WHILE$^{\mathtt{dt}}$ without the constants $\mathtt{dt}$ and ∞.

The language WHILE is much like usual programming languages with a `while` construct, such as **IMP** in the textbook [10]. Its only anomaly is a constant $\mathtt{c}_r$ for any real number r: although unrealistic from the implementation viewpoint, it is fine because WHILE is meant to be a modeling language. Then our target language WHILE$^{\mathtt{dt}}$ is obtained by adding $\mathtt{dt}$ and ∞: they designate an infinitesimal ω^{-1} and an infinite ω.

The relations $>, \leq, \geq$ and $=$ are definable in WHILE$^{\mathtt{dt}}$: $x > y$ as $y < x$; $\leq$ as the negation of $>$; and $=$ as the conjunction of $\leq$ and $\geq$. So are all the Boolean connectives such as $\vee$ and $\Rightarrow$, using $\neg$ and $\wedge$. We emphasize that $\mathtt{dt}$ is by itself a constant and has nothing to do with a variable t. We could have used a more neutral notation like ∂; however the notation $\mathtt{dt}$ turns out to be conveniently intuitive in many examples.

Definition 3.2 (Section of WHILE$^{\mathtt{dt}}$ expression) Let e be an expression of WHILE$^{\mathtt{dt}}$, and $i \in \mathbb{N}$. The *i-th section* of e, denoted by $e|_i$, is obtained by replacing each occurrence of $\mathtt{dt}$ and ∞ in e by the constants $\mathtt{c}_{1/(i+1)}$ and $\mathtt{c}_{i+1}$, respectively. Obviously $e|_i$ is an expression of WHILE.

Example 3.3 (Train control) Our first examples model small fragments of the European Train Control System (ETCS); this is also a leading example in [5]. The following command c_{accel} models a train accelerating at a constant acceleration a, until the time ε is reached. The variable v is for the train's velocity; and z is for its position.

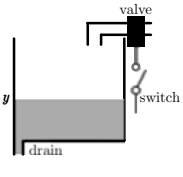
$$\begin{array}{lll} c_{\mathsf{accel}} & :\equiv & \left[\, \mathtt{while}\ t < \varepsilon\ \mathtt{do}\ c_{\mathsf{drive}} \,\right] \qquad \text{where} \\ c_{\mathsf{drive}} & :\equiv & \left[\, t := t + \mathtt{dt}\,;\ v := v + a \cdot \mathtt{dt}\,;\ z := z + v \cdot \mathtt{dt} \,\right] \end{array} \tag{1}$$

The following command $c_{\mathsf{chkAndBrake}}$ models a train that, once its distance from the boundary m gets within the safety distance s, starts braking with the force $b > 0$. However the check if the train is within the safety distance is done only every ε seconds.

$$\begin{array}{lll} c_{\mathsf{chkAndBrake}} & :\equiv & \left[\, \mathtt{while}\ v > 0\ \mathtt{do}\ (\, c_{\mathsf{corr}};\ c_{\mathsf{accel}} \,) \,\right] \qquad \text{where} \\ c_{\mathsf{corr}} & :\equiv & \left[\, t := 0\,;\ \mathtt{if}\ m - z < s\ \mathtt{then}\ a := -b\ \mathtt{else}\ a := 0 \,\right] \end{array} \tag{2}$$

Example 3.4 (Water-level monitor) Our second example is an adaptation from [1]. Compared to the above train example, it exhibits simpler flow-dynamics (the first derivative is already a constant) but more complex jump-dynamics.

There is a water tank with a constant drain (2 cm per second). When the water level y gets lower than 5 cm the switch is turned on, which eventually opens up the valve but only after a time lag of two seconds. While the valve is open, the water level y rises by 1 cm per second. Once y reaches 10 cm the switch is turned off, which will shut the valve but again after a time lag of two seconds.



In the following command c_{water}, the variable x is a timer for a time lag. The `case` construct is an obvious abbreviation of nested `if ... then ... else ...`.

$$c_{\mathsf{water}} :\equiv \left[\begin{array}{l} x := 0;\ y := 1;\ s := 1;\ v := 1; \\ \texttt{while}\ t < t_{\mathsf{max}}\ \texttt{do}\ \ \{ \\ \quad x := x + \mathtt{dt};\quad t := t + \mathtt{dt}; \\ \quad \texttt{if}\ v = 0\ \texttt{then}\ y := y - 2 \cdot \mathtt{dt}\ \texttt{else}\ y := y + \mathtt{dt}; \\ \quad \texttt{case}\ \ \{ s = 0 \wedge v = 0 \wedge y \le 5 : \quad s := 1;\ x := 0; \\ \qquad\qquad s = 1 \wedge v = 0 \wedge x \ge 2 : \quad v := 1; \\ \qquad\qquad s = 1 \wedge v = 1 \wedge 10 \le y : \quad s := 0;\ x := 0; \\ \qquad\qquad s = 0 \wedge v = 1 \wedge x \ge 2 : \quad v := 0; \\ \qquad\qquad \texttt{else} \qquad\qquad\qquad\quad \texttt{skip}\ \ \}\} \end{array}\right] \tag{3}$$

3.2 Denotational Semantics

We follow [10] and interpret a command of WHILE$^{\mathtt{dt}}$ as a transformer on memory states. Our state stores hyperreal numbers such as the infinitesimal $\omega^{-1}{=}[\,(1, \frac{1}{2}, \frac{1}{3}, \dots)\,]$, hence is called a *hyperstate*.

Definition 3.5 (Hyperstate, state) A *hyperstate* $\boldsymbol{\sigma}$ is either $\boldsymbol{\sigma} = \bot$ ("undefined") or a function $\boldsymbol{\sigma} : \mathbf{Var} \to {}^*\mathbb{R}$. A *state* is a standard version of a hyperstate: namely, a *state* σ is either $\sigma = \bot$ or a function $\sigma : \mathbf{Var} \to \mathbb{R}$.

We denote the collection of hyperstates by $\mathbf{HSt}$; that of (standard) states by $\mathbf{St}$.

The definition of (hyper)state as a total function—rather than a partial function with a finite domain—follows [10]. This makes the denotational semantics much simpler. Practically, one can imagine there is a fixed default value (say 0) for any variable.

The following definition is as usual.

Definition 3.6 (State update) Let $\boldsymbol{\sigma} \in \mathbf{HSt}$ be a hyperstate, $x \in \mathbf{Var}$ and $\boldsymbol{r} \in {}^*\mathbb{R}$. We define an *updated hyperstate* $\boldsymbol{\sigma}[x \mapsto \boldsymbol{r}]$ as follows. When $\boldsymbol{\sigma} = \bot$, we set $\bot[x \mapsto \boldsymbol{r}] := \bot$. Otherwise: $\big(\boldsymbol{\sigma}[x \mapsto \boldsymbol{r}]\big)(x) := \boldsymbol{r}$; and for $y \neq x$, $\big(\boldsymbol{\sigma}[x \mapsto \boldsymbol{r}]\big)(y) := \boldsymbol{\sigma}(y)$.

An *updated (standard) state* $\sigma[x \mapsto r]$ is defined analogously.

Definition 3.7 (Sequence representation) Let $(\sigma_i)_{i\in\mathbb{N}}$ be a sequence of (standard) states. It gives rise to a hyperstate—denoted by $[(\sigma_i)_{i\in\mathbb{N}}]$ or simply by $(\sigma_i)_{i\in\mathbb{N}}$—in the following way. We set $(\sigma_i)_{i\in\mathbb{N}} := \bot$ if $\sigma_i = \bot$ for almost all i. Otherwise $\big[\,(\sigma_i)_{i\in\mathbb{N}}\,\big] \neq \bot$ and we set $\big[\,(\sigma_i)_{i\in\mathbb{N}}\,\big](x) := \big[\,(\sigma_i(x))_{i\in\mathbb{N}}\,\big]$, where the latter is the hyperreal represented by the sequence $(\sigma_i(x))_i$ of reals. For $i \in \mathbb{N}$ such that $\sigma_i = \bot$, the value $\sigma_i(x)$ is not defined; in this case we use an arbitrary real number (say 0) for $\sigma_i(x)$. This does not affect the resulting hyperstate since $\sigma_i(x)$ is defined for almost all i.

Let $\boldsymbol{\sigma} \in \mathbf{HSt}$ be a hyperstate, and $(\sigma_i)_{i\in\mathbb{N}}$ be a sequence of states. We say $(\sigma_i)_{i\in\mathbb{N}}$ is a *sequence representation* of $\boldsymbol{\sigma}$ if it gives rise to $\boldsymbol{\sigma}$, that is, $\big[(\sigma_i)_{i\in\mathbb{N}}\big] = \boldsymbol{\sigma}$. In what follows we shall often denote a sequence representation of $\boldsymbol{\sigma}$ by $(\boldsymbol{\sigma}|_i)_{i\in\mathbb{N}}$. We emphasize that given $\boldsymbol{\sigma} \in \mathbf{HSt}$, its sequence representation $(\boldsymbol{\sigma}|_i)_i$ is not unique.

The denotational semantics of WHILE$^{\mathtt{dt}}$ is a straightforward adaptation of the usual semantics of WHILE, except for the `while` clauses where we use sectionwise execution (see Ex. 1.1). As we see later in Lem. 3.10, however, the idea of sectionwise execution extends to the whole language WHILE$^{\mathtt{dt}}$.

Definition 3.8 (Denotational semantics for WHILE$^{\mathtt{dt}}$) For expressions of WHILE$^{\mathtt{dt}}$, their *denotation*

$$\begin{array}{lll} [\![a]\!]: & \mathbf{HSt} \longrightarrow {}^*\mathbb{R} \cup \{\bot\} & \text{for } a \in \mathbf{AExp}, \\ [\![b]\!]: & \mathbf{HSt} \longrightarrow \mathbb{B} \cup \{\bot\} & \text{for } b \in \mathbf{BExp}, \text{ and} \\ [\![c]\!]: & \mathbf{HSt} \longrightarrow \mathbf{HSt} & \text{for } c \in \mathbf{Cmd} \end{array}$$

is defined as follows. Recall that $\bot$ means "undefined" (cf. Def. 3.5); that $\mathbb{B} = \{\mathrm{tt}, \mathrm{ff}\}$ is the set of Boolean truth values; and that ${}^*\mathbb{B} \cong \mathbb{B}$ (Lem. 2.6).

If $\boldsymbol{\sigma} = \bot$, we define $[\![e]\!]\bot := \bot$ for any expression e. If $\boldsymbol{\sigma} \neq \bot$ we define

$$\begin{array}{lll} [\![x]\!]\boldsymbol{\sigma} & := \boldsymbol{\sigma}(x) & [\![\mathtt{c}_r]\!]\boldsymbol{\sigma} := r \quad \text{for each } r \in \mathbb{R} \\ [\![a_1 \mathtt{\ aop\ } a_2]\!]\boldsymbol{\sigma} & := [\![a_1]\!]\boldsymbol{\sigma} \mathtt{\ aop\ } [\![a_2]\!]\boldsymbol{\sigma} & \\ [\![\mathtt{dt}]\!]\boldsymbol{\sigma} & := \omega^{-1} = \big[\,(1, \tfrac{1}{2}, \tfrac{1}{3}, \dots)\,\big] & [\![\infty]\!]\boldsymbol{\sigma} := \omega = \big[\,(1, 2, 3, \dots)\,\big] \end{array}$$

$$\begin{array}{lll} [\![\mathtt{true}]\!]\boldsymbol{\sigma} & := \mathrm{tt} & [\![\mathtt{false}]\!]\boldsymbol{\sigma} := \mathrm{ff} \\ [\![b_1 \wedge b_2]\!]\boldsymbol{\sigma} & := [\![b_1]\!]\boldsymbol{\sigma} \wedge [\![b_2]\!]\boldsymbol{\sigma} & [\![\neg b]\!]\boldsymbol{\sigma} := \neg([\![b]\!]\boldsymbol{\sigma}) \\ [\![a_1 < a_2]\!]\boldsymbol{\sigma} & := [\![a_1]\!]\boldsymbol{\sigma} < [\![a_2]\!]\boldsymbol{\sigma} & \end{array}$$

$$[\![\mathtt{skip}]\!]\boldsymbol{\sigma} := \boldsymbol{\sigma} \qquad [\![x := a]\!]\boldsymbol{\sigma} := \boldsymbol{\sigma}\big[\,x \mapsto [\![a]\!]\boldsymbol{\sigma}\,\big] \qquad [\![c_1; c_2]\!]\boldsymbol{\sigma} := [\![c_2]\!]\big([\![c_1]\!]\boldsymbol{\sigma}\big)$$

$$[\![\mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2]\!]\boldsymbol{\sigma} := \begin{cases} [\![c_1]\!]\boldsymbol{\sigma} & \text{if } [\![b]\!]\boldsymbol{\sigma} = \mathrm{tt} \\ [\![c_2]\!]\boldsymbol{\sigma} & \text{if } [\![b]\!]\boldsymbol{\sigma} = \mathrm{ff} \end{cases}$$

$$[\![\mathtt{while}\ b\ \mathtt{do}\ c]\!]\boldsymbol{\sigma} := \Big([\![\,(\mathtt{while}\ b\ \mathtt{do}\ c)|_i\,]\!](\boldsymbol{\sigma}|_i) \Big)_{i \in \mathbb{N}}\,, \tag{4}$$

where $(\boldsymbol{\sigma}|_i)_{i\in\mathbb{N}}$ is an arbitrary sequence representation of $\boldsymbol{\sigma}$ (Def. 3.7)

Here $\mathtt{aop} \in \{+, -, \times, \hat{}\}$ and $<$ are interpreted on ${}^*\mathbb{R}$ as in Def. 2.4. For each $e \in \mathbf{AExp} \cup \mathbf{BExp}$, we obviously have $[\![e]\!]\boldsymbol{\sigma} = \bot$ if and only if $\boldsymbol{\sigma} = \bot$. It may happen that $[\![c]\!]\boldsymbol{\sigma} = \bot$ with $\boldsymbol{\sigma} \neq \bot$, due to nontermination of `while` loops.

In the semantics of `while` clauses (4), the section $(\mathtt{while}\ b\ \mathtt{do}\ c)|_i$ is a command of WHILE (Def. 3.2); and $\boldsymbol{\sigma}|_i$ is a (standard) state. Thus the state $[\![\,(\mathtt{while}\ b\ \mathtt{do}\ c)|_i\,]\!](\boldsymbol{\sigma}|_i)$ can be defined by the usual semantics of `while` constructs (see e.g. [10]). That is,

$$[\![\mathtt{while}\ b'\ \mathtt{do}\ c']\!]\sigma = \sigma' \quad \overset{\text{def.}}{\Longleftrightarrow} \quad \begin{cases} - \ \sigma = \sigma' = \bot; \\ - \ \text{there exists a finite sequence } \sigma = \sigma_0, \sigma_1, \dots, \sigma_n = \sigma' \text{ such that: } [\![b']\!]\sigma_n = \mathrm{ff}; \text{ and for each } j \in [0, n).\ \big([\![b']\!]\sigma_j = \mathrm{tt}\ \&\ [\![c']\!]\sigma_j = \sigma_{j+1} \big); \text{ or} \\ - \ \text{such a finite sequence does not exist and } \sigma' = \bot. \end{cases} \tag{5}$$

By bundling these up for all i, and regarding it as a hyperstate (Def. 3.7), we obtain the right-hand side of (4). The well-definedness of (4) is proved in Lem. 3.9.

Lemma 3.9 *The semantics of* `while` *clauses (4) is well-defined, being independent of the choice of a sequence representation $(\boldsymbol{\sigma}|_i)_i$ of the hyperstate $\boldsymbol{\sigma}$.* □

In proving the lemma it is crucial that: the set $\{x_1, \dots, x_n\}$ of variables that are relevant to the execution of the command is finite and statically known. This would not be the case with a programming language that allows dynamical creation of fresh variables.

We have chosen not to include the division operator / in WHILE$^{\mathtt{dt}}$; this is to avoid handling of *division by zero* in the semantics, which is cumbersome but seems feasible.

Here is one of our two key lemmas. Its proof is by induction.

Lemma 3.10 (Sectionwise Execution Lemma) *Let e be any expression of* WHILE$^{\mathtt{dt}}$*; $\boldsymbol{\sigma}$ be a hyperstate; and $(\boldsymbol{\sigma}|_i)_{i\in\mathbb{N}}$ be an arbitrary sequence representation of $\boldsymbol{\sigma}$. We have*

$$[\![e]\!]\boldsymbol{\sigma} = \big[\,\big([\![e|_i]\!](\boldsymbol{\sigma}|_i)\big)_{i\in\mathbb{N}}\,\big] \ .$$

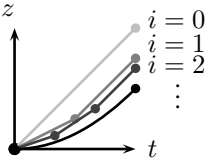
Here the denotational semantics $[\![e|_i]\!]$ of a WHILE *expression $e|_i$ is defined in a usual way (i.e. like in Def. 3.8; for* `while` *clauses see (5)).* □

Example 3.11 Consider c_{accel} in Ex. 3.3. For simplicity we fix the parameters: $c_{\mathsf{accel1}} :\equiv [\,t := 0; \varepsilon := 1; a := 1; v := 0; z := 0; c_{\mathsf{accel}}\,]$. Its i-th section $c_{\mathsf{accel1}}|_i$ has the obvious semantics. For any (standard) state $\sigma \neq \bot$, the real number $([\![c_{\mathsf{accel1}}|_i]\!]\sigma)(z)$—the traveled distance—is easily calculated as

$$\tfrac{1}{i+1}\cdot\tfrac{1}{i+1} + \tfrac{1}{i+1}\cdot\tfrac{2}{i+1} + \cdots + \tfrac{1}{i+1}\cdot\tfrac{i+1}{i+1} = \tfrac{(i+1)(i+2)}{2(i+1)^2} = \tfrac{1}{2}\cdot\tfrac{i+2}{i+1} \ .$$

Therefore by (4), for any hyperstate $\boldsymbol{\sigma} \neq \bot$, the hyperreal $([\![c_{\mathsf{accel1}}]\!]\boldsymbol{\sigma})(z)$ is equal to

$$\big[\,\big(1,\ \tfrac{3}{4},\ \tfrac{2}{3},\ \tfrac{5}{8},\ \ldots,\ \tfrac{1}{2}\cdot\tfrac{i+2}{i+1},\ \ldots\big)\,\big] \ ;$$



this is a hyperreal that is infinitely close to $1/2$.

Much like Ex. 1.1, one way to look at this sectionwise semantics is as an incremental approximation. Here it approximates the solution $z = \frac{1}{2}t^2$ of the differential equation $z'' = 1$, obtained via the Riemann integral. See the above figure.

Remark 3.12 (Denotation of `dt`) We fixed ω^{-1} as the denotation of `dt`. However there are more infinitesimals, such as $(\pi\omega)^{-1} = (\frac{1}{\pi}, \frac{1}{2\pi}, \frac{1}{3\pi}, \ldots)$ with $(\pi\omega)^{-1} < \omega^{-1}$. The choice of `dt`'s denotation does affect the behavior of the following program $c_{\mathsf{nonintegrable}}$:

$$c_{\mathsf{nonintegrable}} \quad :\equiv \quad \big[\, x := 1\,; \quad \texttt{while}\ x \neq 0\ \texttt{do}\ x := x - \texttt{dt}\,\big] \ .$$

When we replace `dt` by $\frac{1}{i+1}$ the program terminates with $x = 0$; hence by our semantics the program yields a non-$\bot$ hyperstate with $x \mapsto 0$. However, replacing `dt` by $\frac{1}{(i+1)\pi}$ with π irrational, the program terminates for no i and it leads to the hyperstate $\bot$.

In fact, indifference to the choice of an infinitesimal value (violated by $c_{\mathsf{nonintegrable}}$) is a typical condition in nonstandard analysis, found e.g. in the characterization of differentiability or Riemann integrability (see [4]). In this sense the program $c_{\mathsf{nonintegrable}}$ is "nonintegrable"; we are yet to see if we can check integrability by syntactic means.

The program $c_{\mathsf{nonintegrable}}$ can be modified into the one with more reasonable behavior, by replacing the guard $x \neq 0$ by $x > 0$. One easily sees that, while different choices of `dt`'s denotation (e.g. ω^{-1} vs. $(\pi\omega)^{-1}$) still lead to different post-hyperstates, the differences lie within infinitesimal gaps. The same is true of all the "realistic" programs that we have looked at.

4 Assertion Language Assn$^{\mathtt{dt}}$

Definition 4.1 (Assn$^{\mathtt{dt}}$, Assn) The syntax of our assertion language Assn$^{\mathtt{dt}}$ is:

$$\begin{array}{lll} \mathbf{AExp} \ni & a & ::= x \mid \mathtt{c}_r \mid a_1 \ \mathtt{aop}\ a_2 \mid \mathtt{dt} \mid \infty \qquad \text{(the same as in While}^{\mathtt{dt}}\text{)} \\ \mathbf{Fml} \ni & A & ::= \mathtt{true} \mid \mathtt{false} \mid A_1 \wedge A_2 \mid \neg A \mid a_1 < a_2 \mid \\ & & \quad \forall x \in {}^*\mathbb{N}.\, A \mid \forall x \in {}^*\mathbb{R}.\, A \qquad \text{where } x \in \mathbf{Var} \end{array}$$

An expression in the family $\mathbf{Fml}$ is called an *(assertion) formula*.

We introduce existential quantifiers as notational conventions: $\exists x \in {}^*\mathbb{D}.\, A :\equiv \neg\forall x \in {}^*\mathbb{D}.\, \neg A$, where $\mathbb{D} \in \{\mathbb{N}, \mathbb{R}\}$.

By Assn we designate the language obtained from Assn$^{\mathtt{dt}}$ by: 1) dropping the constants $\mathtt{dt}, \infty$; and 2) replacing the quantifiers $\forall x \in {}^*\mathbb{N}$ and $\forall x \in {}^*\mathbb{R}$ by $\forall x \in \mathbb{N}$ and $\forall x \in \mathbb{R}$, respectively, i.e. by those which range over standard numbers.

Formulas of Assn$^{\mathtt{dt}}$ are the Boolean expressions of While$^{\mathtt{dt}}$, augmented with quantifiers. The quantifier $\forall x \in {}^*\mathbb{N}$ ranging over hyper-*natural* numbers plays an important role in relative completeness of Hoare$^{\mathtt{dt}}$ (Thm. 5.4).

It is essential that in Assn$^{\mathtt{dt}}$ we have only *hyper*quantifiers like $\forall x \in {}^*\mathbb{R}$ and not *standard* quantifiers like $\forall x \in \mathbb{R}$. The situation is much like with the celebrated *transfer principle* in nonstandard analysis [4, Thm. II.4.5]. There the validity of a standard formula φ is transferred to that of its $*$-transform ${}^*\varphi$; and in ${}^*\varphi$ only hyperquantifiers, and no standard quantifiers, are allowed to occur.

Remark 4.2 (Absence of standard quantifiers) The lack of standard quantifiers does restrict the expressive power of Assn$^{\mathtt{dt}}$. Notably we cannot assert that two hypernumbers x, y are infinitely close, that is, $\forall \varepsilon \in \mathbb{R}.\, (\varepsilon > 0 \Rightarrow -\varepsilon < x - y < \varepsilon)$.[1] However this assertion is arguably unrealistic since, to check it against a physical system, one needs measurements of arbitrarily progressive accuracy. The examples in §6 indicate that Assn$^{\mathtt{dt}}$ is sufficiently expressive for practical verification scenarios, too.

Definition 4.3 (Section of Assn$^{\mathtt{dt}}$ expression) Let e be an expression of Assn$^{\mathtt{dt}}$ (arithmetic or a formula), and $i \in \mathbb{N}$. The *i-th section* of e, denoted by $e|_i$, is obtained by: 1) replacing every occurrence of $\mathtt{dt}$ and ∞ by the constant $\mathtt{c}_{1/(i+1)}$ and $\mathtt{c}_{i+1}$, respectively; and 2) replacing every hyperquantifier $\forall x \in {}^*\mathbb{D}$ by $\forall x \in \mathbb{D}$. Here $\mathbb{D} \in \{\mathbb{N}, \mathbb{R}\}$.

Obviously a section $e|_i$ is an expression of Assn.

Definition 4.4 (Semantics of Assn$^{\mathtt{dt}}$) We define the relation $\boldsymbol{\sigma} \models A$ ("$\boldsymbol{\sigma}$ satisfies A") between a hyperstate $\boldsymbol{\sigma} \in \mathbf{HSt}$ and an Assn$^{\mathtt{dt}}$ formula $A \in \mathbf{Fml}$ as usual.

[1] By replacing $\forall \varepsilon \in \mathbb{R}$ by $\forall \varepsilon \in {}^*\mathbb{R}$ we obtain a legitimate Assn$^{\mathtt{dt}}$ formula, but it is satisfied only when the two hypernumbers x, y are equal.

Namely, if $\boldsymbol{\sigma} = \bot$ we define $\bot \models A$ for each $A \in \mathbf{Fml}$. If $\boldsymbol{\sigma} \neq \bot$, the definition is by the following induction on the construction of A.

$$
\begin{array}{lcll}
\boldsymbol{\sigma} \models \mathtt{true} & & \boldsymbol{\sigma} \not\models \mathtt{false} & \\
\boldsymbol{\sigma} \models A_1 \wedge A_2 & \overset{\text{def.}}{\iff} & \boldsymbol{\sigma} \models A_1 \ \& \ \boldsymbol{\sigma} \models A_2 & \\
\boldsymbol{\sigma} \models \neg A & \overset{\text{def.}}{\iff} & \boldsymbol{\sigma} \not\models A & \\
\boldsymbol{\sigma} \models a_1 < a_2 & \overset{\text{def.}}{\iff} & [\![a_1]\!]\boldsymbol{\sigma} < [\![a_2]\!]\boldsymbol{\sigma} & \text{where } [\![a_i]\!]\boldsymbol{\sigma} \text{ is as defined in Def. 3.8} \\
\boldsymbol{\sigma} \models \forall x \in {}^*\mathbb{D}.\, A & \overset{\text{def.}}{\iff} & \boldsymbol{\sigma}[x \mapsto \boldsymbol{d}] \models A & \text{for each } \boldsymbol{d} \in {}^*\mathbb{D} \qquad (\mathbb{D} \in \{\mathbb{N}, \mathbb{R}\})
\end{array}
$$

Recall that $\boldsymbol{\sigma}[x \mapsto \boldsymbol{d}]$ denotes an updated hyperstate (Def. 3.6).

An ASSN$^{\mathtt{dt}}$ formula $A \in \mathbf{Fml}$ is said to be *valid* if $\boldsymbol{\sigma} \models A$ for any $\boldsymbol{\sigma} \in \mathbf{HSt}$. We denote this by $\models A$. Validity of an ASSN formula is defined similarly.

Lemma 4.5 (Sectionwise Satisfaction Lemma) *Let $A \in \mathbf{Fml}$ be an* ASSN$^{\mathtt{dt}}$ *formula; $\boldsymbol{\sigma}$ be a hyperstate; and $(\boldsymbol{\sigma}|_i)_{i\in\mathbb{N}}$ be an arbitrary sequence representation of $\boldsymbol{\sigma}$. We have*

$$\boldsymbol{\sigma} \models A \quad \textit{if and only if} \quad \big(\, \boldsymbol{\sigma}|_i \models A|_i \quad \textit{for almost every } i \,\big)\ ,$$

where the latter relation $\models$ between standard states and ASSN *formulas is defined in the usual way (i.e. like in Def. 4.4).* □

This is our second key lemma. We note that it fails once we allow standard quantifiers in ASSN$^{\mathtt{dt}}$. For example, let $A :\equiv (\exists y \in \mathbb{R}.\, 0 < y < x)$ and $\boldsymbol{\sigma}$ be a hyperstate such that $\boldsymbol{\sigma}(x) = \omega^{-1}$. Then we have $\boldsymbol{\sigma}|_i \models A|_i$ for every i but $\boldsymbol{\sigma} \not\models A$.

The validity of an ASSN$^{\mathtt{dt}}$ formula A, if A is $(\mathtt{dt}, \infty)$-free, can be reduced to that of an ASSN formula. This is the *transfer principle* for ASSN$^{\mathtt{dt}}$ which we now describe.

Definition 4.6 ($*$-transform) Let A be an ASSN formula. We define its *$*$-transform*, denoted by ${}^*A$, to be the ASSN$^{\mathtt{dt}}$ formula obtained from A by replacing every occurrence of a standard quantifier $\forall x \in \mathbb{D}$ by the corresponding hyperquantifier $\forall x \in {}^*\mathbb{D}$.

It is easy to see that: 1) $({}^*A)|_i \equiv A$ for each ASSN formula A; 2) $A \equiv {}^*(A|_i)$ for each ASSN$^{\mathtt{dt}}$ formula A that is *$(\mathtt{dt}, \infty)$-free*—that is, $\mathtt{dt}$ or ∞ does not occur in it. Then the following is an immediate consequence of Lem. 4.5.

Proposition 4.7 (Transfer principle) *1. For each* ASSN *formula A, $\models A$ iff $\models {}^*A$.*
2. For any $(\mathtt{dt},\infty)$-free ASSN$^{\mathtt{dt}}$ *formula A, the following are equivalent: a) $\models A|_i$ for each $i \in \mathbb{N}$; b) $\models A|_i$ for some $i \in \mathbb{N}$; c) $\models A$.* □

5 Program Logic HOARE$^{\mathtt{dt}}$

We now introduce a Hoare-style program logic HOARE$^{\mathtt{dt}}$ that is devised for the verification of WHILE$^{\mathtt{dt}}$ programs. It derives *Hoare triples* $\{A\}c\{B\}$.

Definition 5.1 (Hoare triple) A *Hoare triple* $\{A\}c\{B\}$ of HOARE$^{\mathtt{dt}}$ is a triple of ASSN$^{\mathtt{dt}}$ formulas A, B and a WHILE$^{\mathtt{dt}}$ command c.

A Hoare triple $\{A\}c\{B\}$ is said to be *valid*—we denote this by $\models \{A\}c\{B\}$—if, for any hyperstate $\boldsymbol{\sigma} \in \mathbf{HSt}$, $\boldsymbol{\sigma} \models A$ implies $[\![c]\!]\boldsymbol{\sigma} \models B$.

As usual a Hoare triple $\{A\}c\{B\}$ asserts *partial correctness*: if the execution of c starting from $\boldsymbol{\sigma}$ does not terminate, we have $[\![c]\!]\boldsymbol{\sigma} = \bot$ hence trivially $[\![c]\!]\boldsymbol{\sigma} \models B$. The formula A in $\{A\}c\{B\}$ is called a *precondition*; B is a *postcondition*.

The rules of HOARE$^{\mathtt{dt}}$ are the same as usual; see e.g. [10].

Definition 5.2 (HOARE$^{\mathtt{dt}}$) The deduction rules of HOARE$^{\mathtt{dt}}$ are as follows.

$$\frac{}{\{A\}\,\mathtt{skip}\,\{A\}}\ (\textsc{Skip}) \qquad \frac{}{\{\,A[a/x]\,\}\,x := a\,\{A\}}\ (\textsc{Assign})$$

$$\frac{\{A\}\,c_1\,\{C\} \quad \{C\}\,c_2\,\{B\}}{\{A\}\,c_1; c_2\,\{B\}}\ (\textsc{Seq}) \qquad \frac{\{A \wedge b\}\,c_1\,\{B\} \quad \{A \wedge \neg b\}\,c_2\,\{B\}}{\{A\}\,\mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2\,\{B\}}\ (\textsc{If})$$

$$\frac{\{A \wedge b\}\,c\,\{A\}}{\{A\}\,\mathtt{while}\ b\ \mathtt{do}\ c\,\{A \wedge \neg b\}}\ (\textsc{While}) \qquad \frac{\models A \Rightarrow A' \quad \{A'\}\,c\,\{B'\} \quad \models B' \Rightarrow B}{\{A\}\,c\,\{B\}}\ (\textsc{Conseq})$$

In the rule (ASSIGN), $A[a/x]$ denotes the capture-avoiding substitution of a for x in A. Recall that $\mathbf{BExp}$ of WHILE$^{\mathtt{dt}}$ is a fragment of $\mathbf{Fml}$ of ASSN$^{\mathtt{dt}}$. Therefore in the rules (IF) and (WHILE), an expression b is an ASSN$^{\mathtt{dt}}$ formula.

We write $\vdash \{A\}c\{B\}$ if the triple $\{A\}c\{B\}$ can be derived using the above rules.

Soundness is a minimal requirement of a logic for verification. The proof makes an essential use of the key "sectionwise" lemmas (Lem. 3.10 and Lem. 4.5).

Theorem 5.3 (Soundness) $\vdash \{A\}c\{B\}$ *implies* $\models \{A\}c\{B\}$. □

We also have a "completeness" result. It is called *relative completeness* [3] since completeness is only modulo the validity of ASSN$^{\mathtt{dt}}$ formulas (namely those in the (CONSEQ) rule); and checking such validity is easily seen to be undecidable. The proof follows the usual method (see e.g. [10, Chap. 7]); namely via explicit description of weakest preconditions.

Theorem 5.4 (Relative completeness) $\models \{A\}c\{B\}$ *implies* $\vdash \{A\}c\{B\}$. □

6 Verification with HOARE$^{\mathtt{dt}}$

We present a couple of examples. Its details as well as some lemmas that aid finding loop invariants will be presented in another venue, due to the lack of space.

Example 6.1 (Water-level monitor) For the program c_{water} in Ex. 3.4, we would like to prove that the water level y stays between 1 cm and 12 cm. It is not hard to see, after some trials, that what we can actually prove is: $\vdash \{\mathtt{true}\}c_{\mathsf{water}}\{1 - 4 \cdot \mathtt{dt} < y < 12 + 2 \cdot \mathtt{dt}\}$. Note that the additional infinitesimal gaps like $4 \cdot \mathtt{dt}$ have no physical meaning. In the proof, we use the following formula A as a loop invariant.

$$\begin{array}{lcl}
A & :\equiv & A_s \wedge A_0 \wedge A_1 \wedge A_2 \wedge A_3 \\
A_s & :\equiv & (s = 0 \vee s = 1) \wedge (v = 0 \vee v = 1) \\
A_0 & :\equiv & s = 1 \wedge v = 1 \quad \Rightarrow \quad 1 - 4 \cdot \mathtt{dt} < y < 10 \\
A_1 & :\equiv & s = 0 \wedge v = 1 \quad \Rightarrow \quad 0 \le x < 2 \quad \wedge \quad 10 \le y < 10 + x + \mathtt{dt} \\
A_2 & :\equiv & s = 0 \wedge v = 0 \quad \Rightarrow \quad 5 < y < 12 + 2 \cdot \mathtt{dt} \\
A_3 & :\equiv & s = 1 \wedge v = 0 \quad \Rightarrow \quad 0 \le x < 2 \quad \wedge \quad 5 - 2x - 2 \cdot \mathtt{dt} < y \le 5
\end{array}$$

Example 6.2 (Train control) Take the program $c_{\mathsf{chkAndBrake}}$ in Ex. 6.2; we aim at the postcondition that the train does not travel beyond the boundary m, that is, $z \leq m$. For simplicity let us first consider $c_{\mathsf{constChkAndBrake}} :\equiv (\varepsilon := \mathtt{dt}\,;\ c_{\mathsf{chkAndBrake}})$. This is the setting where the check is conducted constantly. Indeed we can prove that $\vdash \{v^2 \leq 2b(z-m)\}c_{\mathsf{constChkAndBrake}}\{z \leq m\}$, with a loop invariant $v^2 \leq 2b(z-m)$.

The invariant (and the precondition) $v^2 \leq 2b(z-m)$ is what is derived in [5] by solving a differential equation and then eliminating quantifiers. Using HOARE$^{\mathtt{dt}}$ we can also derive it: roughly speaking, a differential equation in [5] becomes a recurrence relation in our NSA framework. The details and some general lemmas that aid invariant generation are deferred to another venue.

In the general case where $\varepsilon > 0$ is arbitrary, we can prove $\vdash \{v^2 \leq 2b(z-m-v\cdot\varepsilon)\}c_{\mathsf{chkAndBrake}}\{z \leq m\}$ in HOARE$^{\mathtt{dt}}$.

An obvious challenge in verification with HOARE$^{\mathtt{dt}}$ is finding loop invariants. It is tempting—especially with "flow-heavy" systems, i.e. those with predominant flow-dynamics—to assert a differential equation's solution as a loop invariant. This does not work: it is a loop invariant only modulo infinitesimal gaps, a fact not expressible in ASSN$^{\mathtt{dt}}$ (Rem. 4.2). We do not consider this as a serious drawback, for two reasons. Firstly, such "flow-heavy" systems could be studied, after all, from the control theory perspective that is continuous in its origin. The formal verification approach is supposed to show its strength against "jump-heavy" systems, for which differential equations are hardly solvable. Secondly, verification goals are rarely as precise as the solution of a differential equation: we would aim at $z \leq m$ in Ex. 6.2 but not at $z = \frac{1}{2}at^2$.

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theor. Comp. Sci. 138(1), 3–34 (1995)
2. Bliudze, S., Krob, D.: Modelling of complex systems: Systems as dataflow machines. Fundam. Inform. 91(2), 251–274 (2009)
3. Cook, S.A.: Soundness and completeness of an axiom system for program verification. SIAM Journ. Comput. 7(1), 70–90 (1978)
4. Hurd, A.E., Loeb, P.A.: An Introduction to Nonstandard Real Analysis. Academic Press, London (1985)
5. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reasoning 41(2), 143–189 (2008)
6. Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. J. Log. Comput. 20(1), 309–352 (2010)
7. Platzer, A., Quesel, J.-D.: KeYmaera: A hybrid theorem prover for hybrid systems (System description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 171–178. Springer, Heidelberg (2008)
8. Rust, H.: Operational Semantics for Timed Systems: A Non-standard Approach to Uniform Modeling of Timed and Hybrid Systems. LNCS, vol. 3456. Springer, Heidelberg (2005)
9. Suenaga, K., Hasuo, I.: Programming with infinitesimals: A While-language for hybrid system modeling. Extended version with proofs (April 2011)
10. Winskel, G.: The Formal Semantics of Programming Languages. MIT Press, Cambridge (1993)

Model Checking the Quantitative μ-Calculus on Linear Hybrid Systems

Diana Fischer[1] and Łukasz Kaiser[2,⋆]

[1] Mathematische Grundlagen der Informatik, RWTH Aachen University
[2] CNRS & LIAFA, Université Paris Diderot – Paris 7
fischer@logic.rwth-aachen.de, kaiser@liafa.jussieu.fr

Abstract. In this work, we consider the model-checking problem for a quantitative extension of the modal μ-calculus on a class of hybrid systems. Qualitative model checking has been proved decidable and implemented for several classes of systems, but this is not the case for quantitative questions, which arise naturally in this context. Recently, quantitative formalisms that subsume classical temporal logics and additionally allow to measure interesting quantitative phenomena were introduced. We show how a powerful quantitative logic, the quantitative μ-calculus, can be model-checked with arbitrary precision on initialised linear hybrid systems. To this end, we develop new techniques for the discretisation of continuous state spaces based on a special class of strategies in model-checking games and show decidability of a class of counter-reset games that may be of independent interest.

1 Introduction

Modelling discrete-continuous systems by a hybrid of a discrete transition system and continuous variables which evolve according to a set of differential equations is widely accepted in engineering. While model-checking techniques have been applied to verify safety, liveness and other temporal properties of such systems [1,8,9], it is also interesting to infer quantitative values for certain queries. For example, one may not only check that a variable does not exceed a threshold, but also want to compute the maximum value of the variable over all runs.

Thus far, quantitative testing of hybrid systems has only been done by simulation, hence lacking the strong guarantees which can be given by model checking. In recent years, there has been a strong interest to extend classical model-checking techniques and logics to the quantitative setting. Several quantitative temporal logics have been introduced, see e.g. [3,4,6,7,11], together with model-checking algorithms for simple classes of systems, such as finite transition systems with discounts. Still, none of those systems allowed for dynamically changing continuous variables. We present the first model-checking algorithm for a quantitative temporal logic on a class of hybrid systems. The logic we consider, the quantitative μ-calculus [6], is based on a formalism first introduced in [4]. It properly subsumes the standard μ-calculus, thus also CTL and LTL. Therefore

⋆ Authors were supported by DFG AlgoSyn 1298 and ANR 2010 BLAN 0202 02 FREC.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 404–415, 2011.

the present result, namely that it is possible to model-check quantitative μ-calculus on initialised linear hybrid systems, properly generalises a previous result on model-checking LTL on such systems [8,9], which is one of the strongest model-checking results for hybrid systems.

The logic we study allows to express properties involving suprema and infima of values of the considered variables during runs that satisfy various temporal properties, e.g. to answer "what is the maximal temperature on a run during which a safety condition holds". To model-check formulae of the quantitative μ-calculus, we follow the classical parity game-based approach and adapt some of the methods developed in the qualitative case and for timed systems. To our surprise, these methods turned out not to be sufficient and did not easily generalise to the quantitative case. As we will show below, the quantitative systems we study behave in a substantially different way than their qualitative counterparts. We overcome this problem by first working directly with a quantitative equivalence relation, roughly similar to the region graph for timed automata, and finally introducing and solving a new kind of counter-reset games, which may be interesting in their own right.

Organisation. The organisation of this paper follows the reductions needed to model-check a formula φ over a hybrid system $\mathcal{K}$. In Section 2, we introduce the necessary notation, the systems and the logic. Then, we present an appropriate game model in Section 3 and show how to construct a model-checking game $\mathcal{G}$ for the system and the formula. In Section 4, we transform the interval games constructed for arbitrary initialised linear hybrid systems to flat games, where the linear coefficients are always 1. In Section 5, we show how the strategies can be discretised and still lead to a good approximation of the original game. Finally, in Section 6, we solve the obtained parity games with counters.

$$\mathcal{K}, \varphi \rightsquigarrow \text{ model-checking game } \mathcal{G} \rightsquigarrow \text{ flat } \mathcal{G} \rightsquigarrow \text{ counter-reset } \mathcal{G} \rightsquigarrow \text{value.}$$

2 Hybrid Systems and Quantitative Logics

We denote the real and rational numbers and integers extended with both ∞ and $-\infty$ by $\mathbb{R}_\infty$, $\mathbb{Q}_\infty$ and $\mathbb{Z}_\infty$ respectively. We write $\mathcal{I}(\mathbb{Z}_\infty), \mathcal{I}(\mathbb{Q}_\infty)$ and $\mathcal{I}(\mathbb{R}_\infty)$ for all open or closed intervals over $\mathbb{R}_\infty$ with endpoints in $\mathbb{Z}_\infty, \mathbb{Q}_\infty$ and $\mathbb{R}_\infty$. For an interval $I = [i_1, i_2]$, we denote by $q \cdot I$ and $q + I$ the intervals $[q \cdot i_1, q \cdot i_2]$ and $[q + i_1, q + i_2]$, respectively, and do analogously for open intervals. We use the standard meaning of $\lfloor r \rfloor$ and $\lceil r \rceil$, and denote by $\{r\}$ the number $r - \lfloor r \rfloor$ and by $[r]$ the pair $(\lfloor r \rfloor, \lceil r \rceil)$. Hence, when writing $[r] = [s]$, we mean that r and s lie in between the same integers. Note that if $r \in \mathbb{Z}$ then $[r] = [s]$ implies that $r = s$.

Definition 1. *A* linear hybrid system over M variables, $\mathcal{K} = (V, E, \{P_i\}_{i \in J}, \lambda, \delta)$, *is based on a directed graph* (V, E), *consisting of a set of locations* V *and transitions* $E \subseteq V \times V$. *The labelling function* $\lambda : E \to \mathcal{P}_{\text{fin}}(\mathcal{L}_M)$ *assigns to each transition a finite set of labels. For each* i *of the finite index set* J, *the function* $P_i : V \to \mathbb{R}_\infty$ *assigns to each location the value of the static quantitative predicate* P_i. *The function* $\delta : V \to \mathbb{R}^M$ *assigns to each location and variable* x_i *the coefficient* a_i *such that the variable evolves in this location according to the equation*

$\frac{dx_i}{dt} = a_i$. *The set* $\mathcal{L}_M$ *of* transition labels *consists of triples* $l = (I, \overline{C}, R)$, *where the vector* $\overline{C}$ *of length* M *represents the constraints each of the variables need to satisfy for the transition to be allowed, the interval* $I \in \mathcal{I}(\mathbb{R}^{\geq 0}_{\infty})$ *represents the possible period of time that elapses before the transition is taken and the reset set* R *contains the indices of the variables that are reset during the transition, i.e.* $i \in R$ *means that* x_i *is set to zero.*

Note that although we do not explicitly have any invariants in locations, we can simulate this by choosing either the time intervals or variable constraints on the outgoing transitions accordingly. When the values of predicates and labels range over $\mathbb{Q}_\infty$ or $\mathbb{Z}_\infty$ instead of $\mathbb{R}_\infty$ we talk about LHS over $\mathbb{Q}$ and $\mathbb{Z}$.

The *state* of a linear hybrid system $\mathcal{K}$ is a location combined with a valuation of all M variables, $S = V \times \mathbb{R}^M_\infty$. For a state $s = (v, y_1, \ldots, y_M)$ we say that a transition $(v, v') \in E$ is *allowed* by a label $(I, \overline{C}, R) \in \lambda((v, v'))$ if $\overline{y} \in \overline{C}$ (i.e. if $y_i \in C_i$ for all $i = 1, \ldots, M$). We say that a state $s' = (v', y'_1, \ldots, y'_M)$ is a successor of s, denoted $s' \in \mathrm{succ}(s)$, when there is a transition $(v, v') \in E$, allowed by label $(I, \overline{C}, R)$, such that $y'_i = 0$ for all $i \in R$ and there is a $t \in I$ such that $y'_i = y_i + (a_i \cdot t)$ where $a_i = \delta_i(v)$ for all $i \notin R \in \lambda((v, v'))$. A run of a linear hybrid system starting from location v_0 is a sequence of states $s_0, s_1, \ldots$ such that $s_0 = (v_0, 0, \ldots, 0)$ and $s_{i+1} \in \mathrm{succ}(s_i)$ for all i. Given two states s and $s' \in \mathrm{succ}(s)$ and a reset set $R \neq \{1, \ldots, M\}$ we denote by $s' -_R s$ the increase of the non-reset variables that occurred during the transition, i.e. $\frac{y'_i - y_i}{a_i}$ for some $i \notin R$ where $s = (v, \overline{y})$ and $s' = (v', \overline{y}')$.

Definition 2. *A linear hybrid system* $\mathcal{K}$ *is* initialised *if for each* $(v, w) \in E$ *and each variable* x_i *it holds that if* $\delta_i(v) \neq \delta_i(w)$ *then* $i \in R$ *for* $R \in \lambda((v, w))$.

Intuitively, an initialised system cannot store the value of a variable whose evolution rate changes from one location to another.

Example 3. Consider the very simple model of a leaking gas burner depicted in Figure 1. The gas is leaking in location v_0 and not leaking in v_1 and the qualitative predicate L specifies if the leak is detected (and immediately stopped). The system has two variables, x_0 measures the time spent in the leaking location and x_1 the total elapsed time. As both variables are clocks, their coefficients are both one everywhere, i.e. the system is initialised. The time intervals indicate that a gas leak will be detected after at most one time unit and that once the gas is not leaking anymore it can only start to leak again after 30 time units.

2.1 Quantitative μ-Calculus

We use a version of the quantitative μ-calculus presented in [6] but with variables.

Definition 4. *Given fixpoint variables* X_j, *system variables* y_k *and predicates* $\{P_i\}_{i \in J}$, *the formulae of the* quantitative μ-calculus ($Q\mu$) with variables *are given by the grammar:*

$$\varphi \ ::= \ P_i \mid X_j \mid y_k \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi \mid \mu X_j.\varphi \mid \nu X_j.\varphi \,,$$

and in the cases $\mu X_j.\varphi$ *and* $\nu X_j.\varphi$, *the variable* X_j *is required to appear positively in* φ, *i.e. under an even number of negations.*

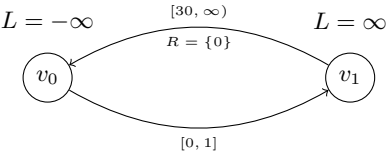


Fig. 1. Leaking gas burner

Let $\mathcal{F} = \{f \ : \ S \to \mathbb{R}_\infty\}$. Given an interpretation $\varepsilon : \mathcal{X} \to \mathcal{F}$, a variable $X \in \mathcal{X}$, and a function $f \in \mathcal{F}$, we denote by $\varepsilon[X \leftarrow f]$ the interpretation ε', such that $\varepsilon'(X) = f$ and $\varepsilon'(X') = \varepsilon(X')$ for all $X' \neq X$.

Definition 5. *Given a linear hybrid system $\mathcal{K} = (V, E, \lambda, \{P_i\}_{i \in J}, \delta)$ and an interpretation ε, a $Q\mu$-formula yields a valuation function $[\![\varphi]\!]^{\mathcal{K}}_\varepsilon : S \to \mathbb{R}_\infty$ defined in the following standard way for a state $s = (v^s, y^s_1, \ldots, y^s_M)$.*

- $[\![P_i]\!]^{\mathcal{K}}_\varepsilon(s) = P_i(v^s)$, $[\![X]\!]^{\mathcal{K}}_\varepsilon(s) = \varepsilon(X)(s)$, *and* $[\![y_i]\!]^{\mathcal{K}}_\varepsilon(s) = y^s_i$, $[\![\neg\varphi]\!]^{\mathcal{K}}_\varepsilon = -[\![\varphi]\!]^{\mathcal{K}}_\varepsilon$
- $[\![\varphi_1 \wedge \varphi_2]\!]^{\mathcal{K}}_\varepsilon = \min\{[\![\varphi_1]\!]^{\mathcal{K}}_\varepsilon, [\![\varphi_2]\!]^{\mathcal{K}}_\varepsilon\}$ *and* $[\![\varphi_1 \vee \varphi_2]\!]^{\mathcal{K}}_\varepsilon = \max\{[\![\varphi_1]\!]^{\mathcal{K}}_\varepsilon, [\![\varphi_2]\!]^{\mathcal{K}}\}$,
- $[\![\Diamond\varphi]\!]^{\mathcal{K}}_\varepsilon(s) = \sup_{s' \in \mathrm{succ}(s)} [\![\varphi]\!]^{\mathcal{K}}_\varepsilon(s')$ *and* $[\![\Box\varphi]\!]^{\mathcal{K}}_\varepsilon(s) = \inf_{s' \in \mathrm{succ}(s)} [\![\varphi]\!]^{\mathcal{K}}_\varepsilon(s')$,
- $[\![\mu X.\varphi]\!]^{\mathcal{K}}_\varepsilon = \inf\{f \in \mathcal{F} : f = [\![\varphi]\!]^{\mathcal{K}}_{\varepsilon[X \leftarrow f]}\}$,
  $[\![\nu X.\varphi]\!]^{\mathcal{K}}_\varepsilon = \sup\{f \in \mathcal{F} : f = [\![\varphi]\!]^{\mathcal{K}}_{\varepsilon[X \leftarrow f]}\}$.

Example 6. The formula $\mu X.(\Diamond X \vee x_1)$ evaluates to the supremum of the values of x_1 on all runs from some initial state: e.g. to ∞ if evaluated on the simple leaking gas burner model. To determine the longest time period of time during which the gas is leaking undetected we use the formula $\mu X.(\Diamond X \vee (x_0 \wedge L))$, which evaluates to 1 on the initial state $(v_0, \overline{0})$ in our example.

For formulae without free variables we write $[\![\varphi]\!]^{\mathcal{K}}$ rather than $[\![\varphi]\!]^{\mathcal{K}}_\varepsilon$. The remainder of this paper is dedicated to the proof of our following main result which shows that $[\![\varphi]\!]^{\mathcal{K}}$ can be approximated with arbitrary precision on initialised linear hybrid systems.

Theorem 7. *Given an initialised linear hybrid system $\mathcal{K}$, a quantitative μ-calculus formula φ and an integer $n > 0$, it is decidable whether $[\![\varphi]\!]^{\mathcal{K}} = \infty$, $[\![\varphi]\!]^{\mathcal{K}} = -\infty$, and else a number $r \in \mathbb{Q}$ can be computed such that $|[\![\varphi]\!]^{\mathcal{K}} - r| < \frac{1}{n}$.*

3 Interval Games

In this section, we define a variant of quantitative parity games suited for model checking $Q\mu$ on linear hybrid systems. This definition is a natural extension of parity games and can be viewed as a compact, finite description for a class of infinite quantitative parity games, which were introduced in [6].

Definition 8. *An* interval parity game *(IPG)* $\mathcal{G} = (V_0, V_1, E, \lambda, \delta, \iota, \Omega)$*, is played on a LHS* (V, E, λ, δ) *and* $V = V_0 \dot\cup V_1$ *is divided into positions of either Player* 0 *or 1. The transition relation* $E \subseteq V \times V$ *describes possible moves in the game which are labelled by the function* $\lambda : E \to \mathcal{P}_{\text{fin}}(\mathcal{L})$ *The function* $\iota : V \to M \times \mathbb{R}_\infty \times \mathbb{R}_\infty$ *assigns to each position the index of a variable and a multiplicative and additive factor, which are used to calculate the payoff if a play ends in this position. The priority function* $\Omega : V \to \{0, \dots, d\}$ *assigns a priority to every position.*

We say that the interval game is over $\mathbb{Q}$ or $\mathbb{Z}$ if both the underlying LHS and all constants in $\iota(v)$ are of the respective kind. Please note that this does not mean that the players have to choose their values from $\mathbb{Q}$ or $\mathbb{Z}$, just that the endpoints of the intervals and constants in the payoffs are in those sets.

A *state* $s = (v, \overline{y}) \in V \times \mathbb{R}_\infty^M$ of an interval game is a position in the game graph together with a variable assignment for all M variables. A state s' is a successor of s if it is a successor in the underlying LHS, i.e. if $s' \in \text{succ}(s)$. We use the functions $\text{loc}(s) = v$ and $\text{var}(s) = \overline{y}, \text{var}_i(s) = y_i$ to access the components of a state. For a real number r, we denote by $r \cdot s = (v, r \cdot \text{var}_0(s), \dots r \cdot \text{var}_M(s))$ and $r + s = (v, r + \text{var}_0(s), \dots r + \text{var}_M(s))$. We call S_i the state set $\{s = (v, \overline{y}) : v \in V_i\}$ where player i has to move and $S = S_0 \dot\cup S_1$.

Intuitively, in a play of an interval parity game, the players choose successors of the current state as long as possible. The outcome $\text{p}(s_0, \dots, s_k)$ of a finite play ending in $s_k = (v, y_1, \dots, y_M)$ if $\iota(v) = (i, a, b)$ is $a \cdot y_i + b$. To improve readability, from now on we will simply write $\iota(v) = a \cdot y_i + b$ instead of $\iota(v) = (i, a, b)$. The outcome of an infinite play depends only on the lowest priority seen infinitely often in positions on the play. We will assign the value $-\infty$ to every infinite play in which the lowest priority seen infinitely often is odd, and ∞ to those, where it is even.

Formally, we use the notion from [6] and define, for an IPG with M variables $\mathcal{G} = (V_0, V_1, E, \lambda, \delta, \iota, \Omega)$, the corresponding infinite quantitative parity game without discounts $\mathcal{G}^* = (V_1 \times \mathbb{R}_\infty^M, V_1 \times \mathbb{R}_\infty^M, E^*, \lambda^*, \Omega^*)$ with $(s, s') \in E^*$ iff s' is a successor of s as above, $\Omega^*(v, \overline{z}) = \Omega(v)$ and $\lambda^*(v, \overline{z}) = \alpha \cdot z_i + \beta$ iff $\iota(v) = \alpha \cdot y_i + \beta$. The notions of plays, strategies, values and determinacy for the IPG $\mathcal{G}$ are defined exactly as the ones for the QPG $\mathcal{G}^*$ in [6].

3.1 Model Checking Games for $Q\mu$

A game $(\mathcal{G}, v)$ is a model checking game for a formula φ and a system $\mathcal{K}, v'$, if the value of the game starting from v is exactly the value of the formula evaluated on $\mathcal{K}$ at v'. In the qualitative case, that means, that φ holds in $\mathcal{K}, v'$ if Player 0 wins in $\mathcal{G}$ from v. For a linear hybrid system $\mathcal{K}$ and a $Q\mu$-formula φ, we construct an IPG $\text{MC}[\mathcal{K}, \varphi]$ which is the model-checking game for φ on $\mathcal{K}$

The full definition of $\text{MC}[\mathcal{K}, \varphi]$ closely follows the construction presented in [6].

Intuitively, the positions are pairs consisting of a sub formula of φ and a location of $\mathcal{K}$. Which player moves at which position depends on the top operator of sub formula. Player 0 moves at disjunctions to a position corresponding to

one of the disjuncts and from $(\Diamond\varphi, v)$ to (φ, w) where $(v, w) \in E^{\mathcal{K}}$, and Player 1 makes analogous moves for conjunctions and $\Box$. From fixed-point variables the play moves back to the defining formula and the priorities of positions depends on the alternation level of fixed points, assigning odd priorities to least fixed points and even priorities to greatest fixed points.

Example 9. We continue our example of the leaking gas burner and present in Figure 2 the model checking game for the previously introduced system and formula. In this interval parity game, ellipses depict positions of Player 0 and rectangles those of Player 1. In this game, all priorities are odd (and therefore omitted), i.e. infinite plays are bad for Player 0. As in the underlying system, there are no constraints on the variables and only in two moves a time unit can be picked by Player 0. In terminal nodes, either the variable x_0 or the predicate L is evaluated for the payoff. The value of the game is 1 (as is the value of the formula on the system starting from either node) and an optimal strategy for Player 0 is picking 1 from $[0, 1]$ and then leaving the cycle where Player 1 is forced to choose between the evaluation of x_0 or L at v_1. Since he is minimising, he will choose to evaluate x_0.

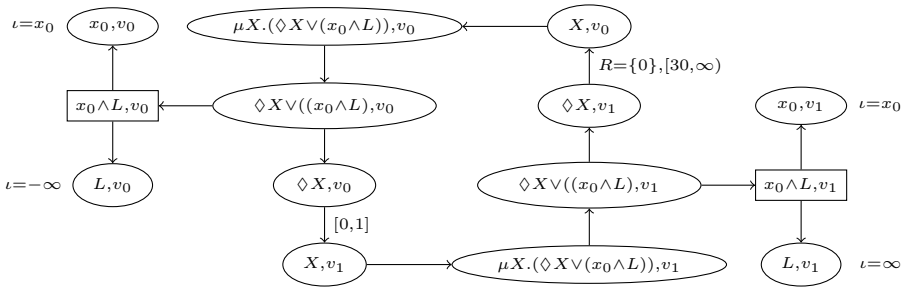


Fig. 2. Model checking game for $\mu X.(\Diamond X \vee (x_0 \wedge L))$ on leaking gas burner

It has been shown in [6] that quantitative parity games of any size are determined and that they are model checking games for $Q\mu$. These results translate to interval parity games and we can conclude the following.

Theorem 10. *Every interval parity game is determined and for every formula φ in $Q\mu$, linear hybrid system $\mathcal{K}$, and a location v of $\mathcal{K}$, it holds that*

$$\mathrm{valMC}[\mathcal{K}, \varphi]((\varphi, v), \overline{0}) = [\![\varphi]\!]^{\mathcal{K}}(v, \overline{0}).$$

4 Basic Properties of Interval Games

At first sight, interval games seem to be very similar to timed games. Simple timed games are solved by playing on the region graph and can thus be discretised. To stress that quantitative payoffs indeed make a difference, we present

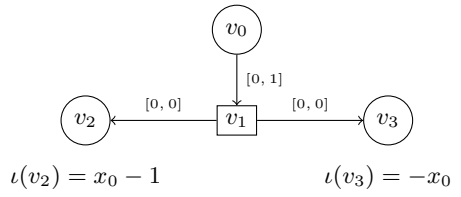


Fig. 3. Game with integer coefficients and non-integer value

in Figure 3 an initialised interval parity game with the interesting property that it is not enough to play integer values, even though the underlying system is over $\mathbb{Z}_\infty$. This simple game contains only one variable (a clock) and has no constraints on the variables in any of the transitions so only the time intervals are shown. Also, as infinite plays are not possible, the priorities are omitted, as well as the indices of non-terminal positions.This game illustrates that it may not be optimal to play integer values since choosing time $\frac{1}{2}$ in the first move is optimal for Player 0. This move guarantees an outcome of $-\frac{1}{2}$ which is equal to the value of the game.

4.1 Flattening Initialised Interval Games

So far, we have considered games where the values of variables can change at different rates during the time spent in locations. In this section, we show that for initialised games it is sufficient to look at easier games where all rates are one, similar to timed games but with more complex payoff rules. We call these games flat and show that for every initialised IPG we can construct a flat IPG with the same value. To do so, we have to consider the regions where the coefficients do not change and rescale the constraints and payoffs accordingly.

Definition 11. *An interval parity game* $\mathcal{G} = (V_0, V_1, E, \lambda, \delta, \iota, \Omega)$ *is* flat *if and only if* $\delta_i(v, \overline{x}) = 1$ *for all* $v \in V$ *and* $i = 1 \dots M$.

Lemma 12. *For each initialised interval parity game* $\mathcal{G}$ *there exists a flat game* $\mathcal{G}'$ *with the same value.*

Consequently, from now on we only consider flat interval parity games and therefore omit the coefficients, as they are all equal to one.

4.2 Multiplying Interval Games

Definition 13. *For a flat IPG* $\mathcal{G} = (V_0, V_1, E, \lambda, \iota, \Omega)$ *and a value* $q \in \mathbb{Q}$*, we denote by* $q \cdot \mathcal{G} = (V, E, \lambda', \iota', \Omega)$ *the IPG where* $\iota'(v) = a \cdot x_i + q \cdot b$ *iff* $\iota(v) = a \cdot x_i + b$ *for all* $v \in V$*, and* $(I', \overline{C'}, R) \in \lambda'((v, w))$ *iff* $(I, \overline{C}, R) \in \lambda((v, w))$ *with* $I' = q \cdot I$ *and* $C'_i = q \cdot C_i$ *for all* $(v, w) \in E$.

Intuitively, this means that all endpoints in the time intervals (open and closed), in the constraints, and all additive values in the payoff function ι are multiplied by q. The values of $q \cdot \mathcal{G}$ are also equal to the values of $\mathcal{G}$ multiplied by q.

Lemma 14. *For every IPG $\mathcal{G}$ over $\mathbb{Q}_\infty$ and $q \in \mathbb{Q}, q \neq 0$ it holds in all states s that $q \cdot \mathrm{val}_{\mathcal{G}}(s) = \mathrm{val}\ q \cdot \mathcal{G}(q \cdot s)$.*

Note that all multiplicative factors in ι are the same in $\mathcal{G}$ and in $q \cdot \mathcal{G}$. Moreover, if we multiply all constants in ι in a game $\mathcal{G}$ (both the multiplicative and the additive ones) by a positive value r, then the value of $\mathcal{G}$ will be multiplied by r, by an analogous argument as above. Thus, if we first take r as the least common multiple of all denominators of multiplicative factors in ι and multiply all ι constants as above, and then take q as the least common multiple of all denominators of endpoints in the intervals and additive factors in the resulting game $\mathcal{G}$ and build $q \cdot \mathcal{G}$, we can conclude the following.

Corollary 15. *For every finite IPG $\mathcal{G}$ over $\mathbb{Q}_\infty$, there exists an IPG $\mathcal{G}'$ over $\mathbb{Z}_\infty$ and $q, r \in \mathbb{Z}$ such that* $\mathrm{val}_{\mathcal{G}}(s) = \frac{\mathrm{val}_{\mathcal{G}'}(q \cdot s)}{q \cdot r}$.

From now on we assume that every IPG we investigate is a flat game over $\mathbb{Z}_\infty$ when not explicitly stated otherwise.

5 Discrete Strategies

Our goal in this section is to show that it suffices to use a simple kind of (almost) discrete strategies to approximate the value of flat interval parity games over $\mathbb{Z}_\infty$. To this end, we define an equivalence relation between states whose variables belong to the same $\mathbb{Z}$ intervals. This equivalence, resembling the standard methods used to build the region graph from timed automata, is a technical tool needed to compare the values of the game in similar states.

Definition 16. *We say that two states s and t in an IPG are equivalent, $s \sim t$, if they are in the same location ($\mathrm{loc}(s) = \mathrm{loc}(t)$) and for all $i, j \in \{1, \ldots, K\}$:*

- $[\mathrm{var}_i(s)] = [\mathrm{var}_i(t)]$, *and*
- *if* $\{\mathrm{var}_i(s)\} \leq \{\mathrm{var}_j(s)\}$ *then* $\{\mathrm{var}_i(t)\} \leq \{\mathrm{var}_j(t)\}$.

Intuitively, all variables lie in the same integer intervals and the order of fractional parts is preserved. In particular, it follows that all integer variables are equal. The following technical lemma allows to shift moves between $\sim$-states.

Lemma 17. *Let s and t be two states in a flat IPG over $\mathbb{Z}$ such that $s \sim t$. If a move from s to s' is allowed by a label $l = (I, \overline{C}, R)$, then there exists a state t', denoted $s'[s/t]$, the move to which from t is allowed by the same label l and*

(1) $t' \sim s'$, and
(2) there is no state $s'' \neq s'$ with (s, s'') allowed by l for which $s''[s/t] = t'$.

Using the lemma above, we can define the notion of shifting play histories. Let $h = t_0 t_1 \ldots t_k$ be a play history such that (t_i, t_{i+1}) is allowed by label l_i and let s_0 be a state, $s_0 \sim t_0$. We say that $s_0 s_1 \ldots s_k$ is a shifted history for h, from the view of player i, if the following conditions are satisfied. For every i if $t_i \in V_i$

then we require that $t_{i+1} = s_{i+1}[s_i/t_i]$. Note that if there is such a s_{i+1} then it is unique by condition (2) of the previous Lemma. If $t_i \in V_{1-i}$ then we require that $s_{i+1} = t_{i+1}[t_i/s_i]$. Note that if there exists a shifted history for h then it is uniquely determined by s_0, and we will denote it (for player i) by $h^i[t_0/s_0]$.

Having defined shifted histories we can shift entire strategies. Given a strategy σ of player i and a state s_0, we define

$$\sigma[s_0/t_0](t_0 \dots t_n) = \begin{cases} \sigma(s_0 \dots s_n)[s_n/t_n] & \text{if } s_0 \dots s_n = (t_0 \dots t_n)^i[s_0/t_0] \text{ exists,} \\ \sigma(t_0 \dots t_n) & \text{otherwise.} \end{cases}$$

This allows to shift whole strategies as stated below.

Lemma 18. *Let $\mathcal{G}$ be a flat IPG over $\mathbb{Z}_\infty$ and let $s_0 \sim t_0$ be two states of $\mathcal{G}$. For any strategies σ of Player 0 and ρ of Player 1 we consider the plays $\pi(\sigma, \rho^1[t_0/s_0], s_0) = s_0 s_1 \dots = \pi_s$ and $\pi(\sigma^0[s_0/t_0], \rho, t_0) = t_0 t_1 \dots = \pi_t$. It holds that either both π_s and π_t are infinite and $\mathrm{p}(\pi_s) = \mathrm{p}(\pi_t)$, or both are finite and of the same length $n+1$ and the last states of these plays s_n and t_n satisfy $s_n \sim t_n$.*

Using the property established above we can finally prove the following.

Lemma 19. *Let $\mathcal{G}$ be a flat IPG over $\mathbb{Z}_\infty$ with the maximal absolute value of the multiplicative factor in payoff functions m, and let $s_0 \sim t_0$. Then $|\mathrm{val}_{\mathcal{G}}(s_0) - \mathrm{val}_{\mathcal{G}}(t_0)| \leq m \cdot |s_0 - t_0|$.*

5.1 Choosing Discrete Moves

We show that for IPGs over $\mathbb{Z}_\infty$, fully general strategies are not necessary. In fact, we can restrict ourselves to discrete strategies and, using this, reduce the games to discrete systems. Intuitively, a discrete strategy keeps the maximal distance of all variable valuations to the closest integer small.

For the proof that there exist good discrete strategies it is convenient to work with the following notion of distance for a state. For $r \in \mathbb{R}$, define

$$\mathrm{d}(r) = \begin{cases} r - \lceil r \rceil & \text{if } |r - \lceil r \rceil| \leq |r - \lfloor r \rfloor|; \\ r - \lfloor r \rfloor & \text{otherwise.} \end{cases}$$

This function gives the distance to the closest integer, except that it is negative if the closest integer is greater than r, i.e. if the fractional part of r is $> \frac{1}{2}$.

For a state s, we use the abbreviation $\mathrm{d}_i(s) = \mathrm{d}(\mathrm{var}_i(s))$. We denote by $\mathrm{d}_l(s) = \min_{i=1\dots k}\{\mathrm{d}_i(s)\}$ and $\mathrm{d}_r(s) = \max_{i=1\dots k}\{\mathrm{d}_i(s)\}$ the smallest and biggest of all values $\mathrm{d}_i(s)$, and additionally we define the total distance as follows.

$$\mathrm{d}^*(s) = \begin{cases} |\mathrm{d}_l(s)| & \text{if } \mathrm{d}_i(s) \leq 0 \text{ for all } i \in \{1, \dots, k\}, \\ \mathrm{d}_r(s) & \text{if } \mathrm{d}_i(s) \geq 0 \text{ for all } i \in \{1, \dots, k\}, \\ |\mathrm{d}_l(s)| + \mathrm{d}_r(s) & \text{otherwise.} \end{cases}$$

First, we will prove that we can always correct a strategy that makes one step which is not ε-discrete. By doing so, we will guarantee that we reach a state with the same location that is allowed by the labelling and the values of the variables only change within the same intervals.

Lemma 20. *Let t be a state with $\mathrm{d}^*(t) \leq \frac{1}{4}$ and s be a successor of t, where (t, s) is allowed by l. Then, for every $0 < \varepsilon < \mathrm{d}^*(t)$, there exists a successor s'_+ of t such that $s \sim s'_+$, (t, s'_+) is allowed by l, and $\mathrm{d}^*(s'_+) \leq \mathrm{d}^*(t) + \varepsilon$.*

Knowing that in one step, the move can always preserve small total distance, we can finally define discrete strategies.

Definition 21. *We call a strategy σ ε-*discrete *if for every $s_{n+1} = \sigma(s_0 \dots s_n)$ it holds that if $\mathrm{d}^*(s_n) \leq \varepsilon$ then $\mathrm{d}^*(s_{n+1}) \leq \mathrm{d}^*(s_n) + \frac{\varepsilon}{2}$.*

Observe that it follows directly from the definition that if $\mathrm{d}^*(s_0) \leq \frac{\varepsilon}{2}$ and both players play discrete strategies, then $\mathrm{d}^*(s_n) \leq \varepsilon(1 - \frac{1}{2^{n+1}})$.

Example 22. To see that decreasing ε in each step is sometimes crucial, consider the game with one variable depicted in Figure 4. In each move Player 0 has to choose a positive value in $(0, 1)$. Player 1 can then decide to continue the play or leave the cycle and end the play with the negative accumulated value, i.e. $-x_0$, as payoff. He cannot infinitely often decide to stay in the cycle as then the payoff would be ∞ as the priority is 0. An ε-optimal strategy for Player 0 as the maximising player is thus to start with $\frac{\varepsilon}{2}$ and decrease in each step.

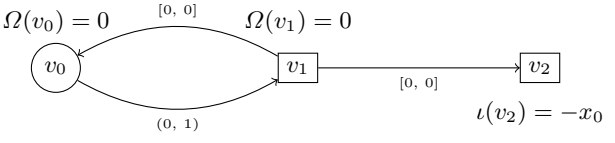


Fig. 4. Game in which the values played must decrease

We now extend the previous lemma to one that allows to shift a whole move.

Lemma 23. *Let s be a state and s' a successor of s, where (s, s') is allowed by l Let t be a state with $\mathrm{d}^*(t) \leq \frac{1}{4}$, such that $s \sim t$. Then for every $\varepsilon > 0$ there exists a successor t' of t allowed by l such that $t \sim t'$ and $\mathrm{d}^*(t') \leq \mathrm{d}^*(t) + \varepsilon$.*

We can conclude that discrete strategies allow to approximate game values.

Lemma 24. *For every strategy σ of Player i in $\mathcal{G}$, there is a discrete strategy σ_d, such that for any starting state s_0 and discrete strategy ρ of the other player, if $\pi(\sigma, \rho, s_0) = s_0, s_1, \dots$ and $\pi(\sigma_d, \rho, s_0) = s'_0, s'_1, \dots$, then $s_i \sim s'_i$ for all i.*

Proposition 25. *Let $\mathcal{G}$ be a flat interval parity game. Let Γ_i be the set of all strategies for player i and Δ_i the set of all discrete strategies for player i and m be the highest value that occurs as a multiplicative factor in ι. Then it holds, for every starting state s, that*

$$|\sup_{\sigma \in \Gamma_0} \inf_{\rho \in \Gamma_1} \mathrm{p}(\pi(\sigma, \rho, s)) - \sup_{\sigma \in \Delta_0} \inf_{\rho \in \Delta_1} \mathrm{p}(\pi(\sigma, \rho, s))| \leq m.$$

6 Counter-Reset Games

By the above Proposition 25, we can restrict both players to use ε-discrete strategies to approximate the value of a flat interval game up to the maximal multiplicative factor m. Multiplying the game by any number q does not change the multiplicative factors in ι but multiplies the value of the game by q. Thus, to approximate the value of $\mathcal{G}$ up to $\frac{1}{n}$ it suffices to play ε-discrete strategies in $n \cdot m \cdot \mathcal{G}$. When players use only discrete strategies, the chosen values remain close to integers (possibly being up to ε bigger or smaller). The fact whether the value is bigger, equal or smaller than an integer can be stored in the state, as well as whether the value of a variable is smaller than any of the (non-infinite) bounds in constraint intervals or bigger than all of them. This way, we can eliminate both ε's and constraints and are left with the following games.

Definition 26. *A* counter-reset *game is a flat interval parity game in which in each label $l = (I, \overline{C}, R)$ the constraints $\overline{C}$ are trivially true and the interval I is either $[0,0]$ or $[1,1]$, i.e. either all variables are incremented by 1 or all are left intact. A* generalised counter-reset game *is one in which each variable separately is assigned to be incremented or to be left intact in each move.*

Lemma 27. *Let $\mathcal{G}$ be an IPG over $\mathbb{Z}_\infty$ with maximal absolute value of the multiplicative factor in ι equal to m. For each $n \in \mathbb{N}$ there exists a counter-reset game $\mathcal{G}'_n$ such that for all states s in which all variables are integers:*

$$|\mathrm{val}\mathcal{G}(s) - \frac{\mathrm{val}\mathcal{G}'_n(n \cdot m \cdot s)}{n \cdot m}| \leq \frac{1}{n}.$$

We solve (even the generalised) counter-reset games in a similar way as classical parity games. We start with games of finite duration and observe that there is a simple symbolic representation for the payoffs, in terms of min-max functions, which can be achieved by the players. Then, we use the unfolding theorem from [6] and compute fixed-points over this representation to solve these parity games. To exploit this fixed-point computation algorithmically, it is necessary to show that a fixed-point in the chosen symbolic representation will be reached in a finite number of steps. To this end, we use a form of Dickson's Lemma applied to linearly controlled functions [10,5]. This allows us to show convergence of the fixed-points and thus exactly calculate the value of a counter-reset game.

Proposition 28. *Given a generalised counter-reset game $\mathcal{G}$ and a state s in which all counters are integers, one can compute* $\mathrm{val}\mathcal{G}(s)$.

7 Conclusions and Future Work

We conclude by completing the proof of our main Theorem 7. We first observe that, by Theorem 10, evaluating a $Q\mu$-formula on a system is equivalent to calculating the value of the corresponding model-checking game. We can then

turn this game into a flat one by Lemma 12 and then into one over $\mathbb{Z}_\infty$ by Corollary 15. By Lemma 27 the value of such a game can be approximated arbitrarily precise by counter-reset games, which we can solve by Proposition 28.

All together, we proved that it is possible to approximate the values of quantitative μ-calculus formulae on initialised linear hybrid systems with arbitrary precision. Unfortunately, we cannot give complexity bounds for our procedure, even though we believe that the algorithm we presented is elementary (and related to questions about cost-MSO, a recently studied logic [2]). Two immediate problems remain open: (1) can the exact value of $[\![\varphi]\!]^{\mathcal{K}}$ be computed? (2) what is the complexity of such a computation or its approximation? Even with further research needed to answer these questions, our result lays the foundation for using temporal logics for the quantitative verification of hybrid systems.

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science 138(1), 3–34 (1995)
2. Colcombet, T., Löding, C.: Regular cost functions over finite trees. In: LICS, pp. 70–79. IEEE Computer Society, Los Alamitos (2010)
3. de Alfaro, L.: Quantitative verification and control via the mu-calculus. In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 103–127. Springer, Heidelberg (2003)
4. de Alfaro, L., Faella, M., Stoelinga, M.: Linear and Branching Metrics for Quantitative Transition Systems. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 97–109. Springer, Heidelberg (2004)
5. Figueira, D., Figueira, S., Schmitz, S., Schnoebelen, P.: Ackermann and primitive-recursive bounds with dickson's lemma. CoRR, abs/1007.2989 (2010)
6. Fischer, D., Grädel, E., Kaiser, L.: Model checking games for the quantitative μ-calculus. Theory Comput. Syst. 47(3), 696–719 (2010)
7. Gawlitza, T., Seidl, H.: Computing game values for crash games. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 177–191. Springer, Heidelberg (2007)
8. Henzinger, T.A., Horowitz, B., Majumdar, R.: Rectangular hybrid games. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 320–335. Springer, Heidelberg (1999)
9. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? In: Proceedings of STOC 1995, pp. 373–382. ACM, New York (1995)
10. McAloon, K.: Petri nets and large finite sets. Theoretical Computer Science 32, 173–183 (1984)
11. McIver, A., Morgan, C.: Results on the quantitative μ-calculus qMμ. ACM Trans. Comput. Log. 8(1) (2007)

On Reachability for Hybrid Automata over Bounded Time*

Thomas Brihaye[1], Laurent Doyen[2], Gilles Geeraerts[3], Joël Ouaknine[4], Jean-François Raskin[3], and James Worrell[4]

[1] Université de Mons, Belgium
[2] LSV, ENS Cachan & CNRS, France
[3] Université Libre de Bruxelles, Belgium
[4] Oxford University Computing Laboratory, UK

Abstract. This paper investigates the time-bounded version of the reachability problem for hybrid automata. This problem asks whether a given hybrid automaton can reach a given target location within $\mathbf{T}$ time units, where $\mathbf{T}$ is a constant rational value. We show that, in contrast to the classical (unbounded) reachability problem, the timed-bounded version is *decidable* for rectangular hybrid automata provided only non-negative rates are allowed. This class of systems is of practical interest and subsumes, among others, the class of stopwatch automata. We also show that the problem becomes undecidable if either diagonal constraints or both negative and positive rates are allowed.

1 Introduction

The formalism of hybrid automata [1] is a well-established model for hybrid systems whereby a digital controller is embedded within a physical environment. The state of a hybrid system changes both through discrete transitions of the controller, and continuous evolutions of the environment. The discrete state of the system is encoded by the *location* ℓ of the automaton, and the continuous state is encoded by *real-valued variables* X evolving according to dynamical laws constraining the first derivative $\dot{X}$ of the variables. Hybrid automata have proved useful in many applications, and their analysis is supported by several tools [6,5].

A central problem in hybrid-system verification is the *reachability problem* which is to decide if there exists an execution from a given initial location ℓ to a given goal location ℓ'. While the reachability problem is undecidable for simple classes of hybrid

* Work supported by the projects: (i) QUASIMODO (FP7- ICT-STREP-214755), Quasimodo: "Quantitative System Properties in Model-Driven-Design of Embedded", `http://www.quasimodo.aau.dk/`, (ii) GASICS (ESF-EUROCORES LogiCCC), Gasics: "Games for Analysis and Synthesis of Interactive Computational Systems", `http://www.ulb.ac.be/di/gasics/`, (iii) Moves: "Fundamental Issues in Modelling, Verification and Evolution of Software", `http://moves.ulb.ac.be`, a PAI program funded by the Federal Belgian Government, (iv) the ARC project AUWB-2010–10/15-UMONS-3, (v) the FRFC project 2.4515.11 and (vi) a grant from the National Bank of Belgium.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 416–427, 2011.

automata (such as linear hybrid automata [1]), the decidability frontier of this problem is sharply understood [7,8]. For example, the reachability problem is decidable for the class of initialized rectangular automata where (i) the flow constraints, guards, invariants and discrete updates are defined by rectangular constraints of the form $a \leq \dot{x} \leq b$ or $c \leq x \leq d$ (where a, b, c, d are rational constants), and (ii) whenever the flow constraint of a variable x changes between two locations ℓ and ℓ', then x is reset along the transition from ℓ to ℓ'. Of particular interest is the class of timed automata which is a special class of initialized rectangular automata [2].

In recent years, it has been observed that new decidability results can be obtained in the setting of time-bounded verification of real-time systems [10,11]. Given a time bound $\mathbf{T} \in \mathbb{N}$, the time-bounded verification problems consider only traces with duration at most $\mathbf{T}$. Note that due to the density of time, the number of discrete transitions may still be unbounded. Several verification problems for timed automata and real-time temporal logics turn out to be decidable in the time-bounded framework (such as the language-inclusion problem for timed automata [10]), or to be of lower complexity (such as the model-checking problem for MTL [11]). The theory of time-bounded verification is therefore expected to be more robust and better-behaved in the case of hybrid automata as well.

Following this line of research, we revisit the reachability problem for hybrid automata with time-bounded traces. The *time-bounded reachability problem* for hybrid automata is to decide, given a time bound $\mathbf{T} \in \mathbb{N}$, if there exists an execution of duration less than $\mathbf{T}$ from a given initial location ℓ to a given goal location ℓ'. We study the frontier between decidability and undecidability for this problem and show how bounding time alters matters with respect to the classical reachability problem. In this paper, we establish the following results. First, we show that the time-bounded reachability problem is *decidable* for non-initialized rectangular automata when only positive rates are allowed[1]. The proof of this fact is technical and, contrary to most decidability results in the field, does not rely on showing the existence of an underlying finite (bi)simulation quotient. We study the properties of time-bounded runs and show that if a location is reachable within $\mathbf{T}$ time units, then it is reachable by a timed run in which the number of discrete transitions can be bounded. This in turn allows us to reduce the time-bounded reachability problem to the satisfiability of a formula in the first-order theory of real addition, decidable in EXPSPACE [4].

Second, we show that the time-bounded reachability problem is *undecidable* for non-initialized rectangular hybrid automata if both positive and negative rates are allowed. Third, we show that the time-bounded reachability problem is *undecidable* for initialized rectangular hybrid automata with positive singular flows if diagonal constraints in guards are allowed. These two undecidability results allow to precisely characterize the boundary between decidability and undecidability.

The undecidability results are obtained by reductions from the halting problem for two-counter machines. We present novel encodings of the execution of two-counter machines that fit into time-bounded executions of hybrid automata with either negative rates, or diagonal constraints.

[1] This class is interesting from a practical point of view as it includes, among others, the class of stopwatch automata [3], for which unbounded reachability is undecidable.

Remark Due to lack of space, most of the proofs are omitted or sketched in the present version. A complete version with all the proofs is available for download on `arxiv.org`.

2 Definitions

Let $\mathcal{I}$ be the set of intervals of real numbers with endpoints in $\mathbb{Z} \cup \{-\infty, +\infty\}$. Let X be a set of continuous variables, and let $X' = \{x' \mid x \in X\}$ and $\dot{X} = \{\dot{x} \mid x \in X\}$ be the set of primed and dotted variables, corresponding respectively to variable updates and first derivatives. A *rectangular constraint* over X is an expression of the form $x \in I$ where x belongs to X and I to $\mathcal{I}$. A *diagonal constraint* over X is a constraint of the form $x - y \sim c$ where x, y belong to X, c to $\mathbb{Z}$, and $\sim$ is in $\{<, \leq, =, \geq, >\}$. Finite conjunctions of diagonal and rectangular constraints over X are called *guards*, over $\dot{X}$ they are called *rate constraints*, and over $X \cup X'$ they are called *update constraints*. A guard or rate constraint is *rectangular* if all its constraints are rectangular. An update constraint is *rectangular* if all its constraints are either rectangular or of the form $x = x'$. We denote by $\mathcal{G}(X), \mathcal{R}(X), \mathcal{U}(X)$ respectively the sets of guards, rate constraints, and update constraints over X.

Linear hybrid automata. A *linear hybrid automaton* (LHA) is a tuple $\mathcal{H} = (X, \mathrm{Loc}, \mathrm{Edges}, \mathrm{Rates}, \mathrm{Inv}, \mathrm{Init})$ where $X = \{x_1, \ldots, x_{|X|}\}$ is a finite set of continuous *variables*; Loc is a finite set of *locations*; $\mathrm{Edges} \subseteq \mathrm{Loc} \times \mathcal{G}(X) \times \mathcal{U}(X) \times \mathrm{Loc}$ is a finite set of *edges*; $\mathrm{Rates} : \mathrm{Loc} \mapsto \mathcal{R}(X)$ assigns to each location a constraint on the *possible variable rates*; $\mathrm{Inv} : \mathrm{Loc} \mapsto \mathcal{G}(X)$ assigns an *invariant* to each location; and $\mathrm{Init} \in \mathrm{Loc}$ is an *initial location*. For an edge $e = (\ell, g, r, \ell')$, we denote by $\mathsf{src}(e)$ and $\mathsf{trg}(e)$ the location ℓ and ℓ' respectively, g is called the *guard* of e and r is the *update* (or *reset*) of e. In the sequel, we denote by rmax the maximal constant occurring in the constraints of $\{\mathrm{Rates}(\ell) \mid \ell \in \mathrm{Loc}\}$

A LHA $\mathcal{H}$ is *singular* if for all locations ℓ and for all variables x of $\mathcal{H}$, the only constraint over $\dot{x}$ in $\mathrm{Rates}(\ell)$ is of the form $\dot{x} \in I$ where I is a singular interval; it is *fixed rate* if for all variables x of $\mathcal{H}$ there exists $I_x \in \mathcal{I}$ such that for all locations ℓ of $\mathcal{H}$, the only constraint on $\dot{x}$ in $\mathrm{Rates}(\ell)$ is the constraint $\dot{x} \in I_x$. It is *multirate* if it is not fixed rate. It is *non-negative rate* if for all variables x, for all locations ℓ, the constraint $\mathrm{Rates}(\ell)$ implies that $\dot{x}$ must be non-negative.

Rectangular hybrid automata. A *rectangular hybrid automaton* (RHA) is a linear hybrid automaton in which all guards, rates, and invariants are rectangular. In this case, we view each reset r as a function $X' \mapsto \mathcal{I} \cup \{\bot\}$ that associates to each variable $x \in X$ either an interval of possible reset values $r(x)$, or $\bot$ when the value of the variable x remains unchanged along the transition. When it is the case that $r(x)$ is either $\bot$ or a singular interval for each x, we say that r is *deterministic*. In the case of RHA, we can also view rate constraints as functions $\mathrm{Rates} : \mathrm{Loc} \times X \to \mathcal{I}$ that associate to each location ℓ and each variable x an interval of possible rates $\mathrm{Rates}(\ell)(x)$. A rectangular hybrid automaton $\mathcal{H}$ is *initialized* if for every edge (ℓ, g, r, ℓ') of $\mathcal{H}$, for every $x \in X$, if $\mathrm{Rates}(\ell)(x) \neq \mathrm{Rates}(\ell')(x)$ then $r(x) \neq \bot$, i.e., every variable whose rate constraint is changed must be reset.

LHA semantics. A *valuation* of a set of variables X is a function $\nu : X \mapsto \mathbb{R}$. We further denote by $\mathbf{0}$ the valuation that assigns 0 to each variable.

Given an LHA $\mathcal{H} = (X, \mathrm{Loc}, \mathrm{Edges}, \mathrm{Rates}, \mathrm{Inv}, \mathrm{Init}, X)$, a *state* of $\mathcal{H}$ is a pair (ℓ, ν), where $\ell \in \mathrm{Loc}$ and ν is a valuation of X. The semantics of $\mathcal{H}$ is defined as follows. Given a state $s = (\ell, \nu)$ of $\mathcal{H}$, an *edge step* $(\ell, \nu) \xrightarrow{e} (\ell', \nu')$ can occur and change the state to (ℓ', ν') if $e = (\ell, g, r, \ell') \in \mathrm{Edges}$, $\nu \models g$, $\nu'(x) = \nu(x)$ for all x s.t. $r(x) = \bot$, and $\nu'(x) \in r(x)$ for all x s.t. $r(x) \neq \bot$; given a time delay $t \in \mathbb{R}^+$, a *continuous time step* $(\ell, \nu) \xrightarrow{t} (\ell, \nu')$ can occur and change the state to (ℓ, ν') if there exists a vector $r = (r_1, \dots r_{|X|})$ such that $r \models \mathrm{Rates}(\ell)$, $\nu' = \nu + (r \cdot t)$, and $\nu + (r \cdot t') \models \mathrm{Inv}(\ell)$ for all $0 \leq t' \leq t$.

A *path* in $\mathcal{H}$ is a finite sequence $e_1, e_2, \dots, e_n$ of edges such that $\mathsf{trg}(e_i) = \mathsf{src}(e_{i+1})$ for all $1 \leq i \leq n-1$. A *cycle* is a path $e_1, e_2, \dots, e_n$ such that $\mathsf{trg}(e_n) = \mathsf{src}(e_1)$. A cycle $e_1, e_2, \dots, e_n$ is *simple* if $\mathsf{src}(e_i) \neq \mathsf{src}(e_j)$ for all $i \neq j$. A *timed path* of $\mathcal{H}$ is a finite sequence of the form $\pi = (t_1, e_1), (t_2, e_2), \dots, (t_n, e_n)$, such that $e_1, \dots, e_n$ is a path in $\mathcal{H}$ and $t_i \in \mathbb{R}^+$ for all $0 \leq i \leq n$. We lift the notions of cycle and simple cycle to the timed case accordingly. Given a timed path $\pi = (t_1, e_1), (t_2, e_2), \dots, (t_n, e_n)$, we denote by $\pi[i:j]$ (with $1 \leq i \leq j \leq n$) the timed path $(t_i, e_i), \dots, (t_j, e_j)$.

A *run* in $\mathcal{H}$ is a sequence $s_0, (t_0, e_0), s_1, (t_1, e_1), \dots, (t_{n-1}, e_{n-1}), s_n$ such that:

- $(t_0, e_0), (t_1, e_1), \dots, (t_{n-1}, e_{n-1})$ is a timed path in $\mathcal{H}$, and
- for all $1 \leq i < n$, there exists a state s'_i of $\mathcal{H}$ with $s_i \xrightarrow{t_i} s'_i \xrightarrow{e_i} s_{i+1}$.

Given a run $\rho = s_0, (t_0, e_0), \dots, s_n$, let $\mathsf{first}(\rho) = s_0 = (\ell_0, \nu_0)$, $\mathsf{last}(\rho) = s_n$, $\mathsf{duration}(\rho) = \sum_{i=1}^{n-1} t_i$, and $|\rho| = n + 1$. We say that ρ is (i) *strict* if $t_i > 0$ for all $1 \leq i \leq n-1$; (ii) *k-variable-bounded* (for $k \in \mathbb{N}$) if $\nu_0(x) \leq k$ for all $x \in X$, and $s_i \xrightarrow{t_i} (\ell_i, \nu_i)$ implies that $\nu_i(x) \leq k$ for all $0 \leq i \leq n$; (iii) **T**-*time-bounded* (for $\mathbf{T} \in \mathbb{N}$) if $\mathsf{duration}(\rho) \leq \mathbf{T}$.

Note that a unique timed path $\mathsf{TPath}(\rho) = (t_0, e_0), (t_1, e_1), \dots, (t_{n-1}, e_{n-1})$, is associated to each run $\rho = s_0, (t_0, e_0), s_1, \dots, (t_{n-1}, e_{n-1}), s_n$. Hence, we sometimes abuse notation and denote a run ρ with $\mathsf{first}(\rho) = s_0$, $\mathsf{last}(\rho) = s$ and $\mathsf{TPath}(\rho) = \pi$ by $s_0 \xrightarrow{\pi} s$. The converse however is not true: given a timed path π and an initial state s_0, it could be impossible to build a run starting from s_0 and following π because some guards or invariants along π might be violated. However, if such a run exists it is necessarily unique *when the automaton is singular and all resets are deterministic*. In that case, we denote by $\mathsf{Run}(s_0, \pi)$ the function that returns the unique run ρ such that $\mathsf{first}(\rho) = s_0$ and $\mathsf{TPath}(\rho) = \pi$ if it exists, and $\bot$ otherwise.

Time-bounded reachability problem for LHA. While the reachability problem asks to decide the existence of any timed run that reaches a given goal location, we are only interested in runs having bounded duration.

Problem 1 (Time-bounded reachability problem). Given an LHA $\mathcal{H} = (X, \mathrm{Loc}, \mathrm{Edges}, \mathrm{Rates}, \mathrm{Inv}, \mathrm{Init})$, a location $\mathrm{Goal} \in \mathrm{Loc}$ and a time bound $\mathbf{T} \in \mathbb{N}$, the *time-bounded reachability problem* is to decide whether there exists a finite run $\rho = (\mathrm{Init}, \mathbf{0}) \xrightarrow{\pi} (\mathrm{Goal}, \cdot)$ of $\mathcal{H}$ with $\mathsf{duration}(\rho) \leq \mathbf{T}$.

In the following table, we summarize the known facts regarding decidability of the reachability problem for LHA, along with the results on time-bounded reachability that we prove in the rest of this paper. Note that decidability for initialized rectangular hybrid automata (IHRA) follows directly from [7]. We show decidability for (non-initialized) RHA that only have non-negative rates in Section 3. The undecidability of the time-bounded reachability problem for RHA and LHA is not a consequence of the known results from the literature and require new proofs that are given in Section 4.

HA classes	Reachability	Time-Bounded Reachability
LHA	U [1]	**U** (see Section 4)
RHA	U [7]	**U** (see Section 4)
non-negative rates RHA	U [7]	**D** (see Section 3)
IRHA	D [7]	D [7]

3 Decidability for RHA with Non-negative Rates

In this section, we prove that the time-bounded reachability problem is *decidable* for the class of (non-initialized) *rectangular* hybrid automata having *non-negative rates*, while it is *undecidable* for this class in the classical (unbounded) case [7]. Note that this class is interesting in practice since it contains, among others, the important class of *stopwatch automata*, a significant subset of LHA that has several useful applications [3]. We obtain decidability by showing that for RHA with non-negative rates, a goal location is reachable within $\mathbf{T}$ time units iff there exists a witness run of that automaton which reaches the goal (within $\mathbf{T}$ time units) by a run ρ of length $|\rho| \leq K_{\mathbf{T}}^{\mathcal{H}}$ where $K_{\mathbf{T}}^{\mathcal{H}}$ is a parameter that depends on $\mathbf{T}$ and on the size of the automaton $\mathcal{H}$. Time-bounded reachability can thus be reduced to the satisfiability of a formula in the first order theory of the reals encoding the existence of runs of length at most $K_{\mathbf{T}}^{\mathcal{H}}$ and reaching Goal.

For simplicity of the proofs, we consider RHA with the following restrictions: (i) the guards *do not contain strict inequalities*, and (ii) the rates are *singular*. We argue at the end of this section that these restrictions can be made without loss of generality. Then, in order to further simplify the presentation, we show how to syntactically simplify the automaton while preserving the time-bounded reachability properties. The details of the constructions can be found in the appendix.

Proposition 1. *Let $\mathcal{H}$ be a singular RHA with non-negative rates and without strict inequalities, and let* Goal *be a location of $\mathcal{H}$. We can build a hybrid automaton $\mathcal{H}'$ with the following the properties:*

- H_1 *$\mathcal{H}'$ is a singular RHA with non-negative rates*
- H_2 *$\mathcal{H}'$ contains only deterministic resets*
- H_3 *for every edge (ℓ, g, r, ℓ') of $\mathcal{H}'$, g is either* **true** *or of the form $x_1 = 1 \wedge x_2 = 1 \wedge \cdots \wedge x_k = 1$, and $r \equiv x_1' = 0 \wedge \cdots \wedge x_k' = 0$.*

and a set of locations S of $\mathcal{H}'$ such that $\mathcal{H}$ admits a $\mathbf{T}$-time bounded run reaching Goal *iff $\mathcal{H}'$ admits a strict 1-variable-bounded, and $\mathbf{T}$-time bounded run reaching S.*

Proof (Sketch). The proof exposes three transformations that we apply to $\mathcal{H}$ in order to obtain $\mathcal{H}'$. The first transformation turns $\mathcal{H}$ into $\mathsf{DetReset}(\mathcal{H})$, containing deterministic resets only. The idea is to replace (non-deterministic) resets in $\mathcal{H}$ with resets to 0 in $\mathsf{DetReset}(\mathcal{H})$ and to compensate by suitably altering the guards of subsequent transitions in $\mathsf{DetReset}(\mathcal{H})$. To achieve this, locations in $\mathsf{DetReset}(\mathcal{H})$ are elements of the form (ℓ, ρ), where ℓ is a location of $\mathcal{H}$ and ρ associates an interval to each variable, where $\rho(j)$ represents the interval in which variable x_j was last reset.

With the second transformation, we can restrict our analysis to runs where the variables are bounded by 1. The idea is to encode the integer parts of the variables in the locations, and to adapt the guards and the resets. Let $\mathcal{H}'$ be an RHA obtained from the first step, with maximal constant cmax. We build $\mathsf{CBound}(\mathcal{H}')$ whose locations are of the form $(\ell, \mathbf{i})$, where ℓ is a location of $\mathcal{H}'$, and $\mathbf{i}$ is a function that associates a value from $\{0, \ldots, \mathrm{cmax}\}$ to each variable. Intuitively, $\mathbf{i}(j)$ represents the integer part of x_j in the original run of $\mathcal{H}'$, whereas the fractional part is tracked by x_j (hence all the variables stay in the interval $[0, 1]$). Guards and resets are adapted consequently.

The third and last construction allows to consider only runs where time is strictly increasing. We describe it briefly assuming all the invariants are $\mathbf{true}$ to avoid technicalities. Consider a sequence of edges and null time delays of the form $e_1, 0, e_2, 0, \ldots, 0, e_n$ (remark that this sequence ends and starts with an edge). Since all time delays are null, the only effect of firing such as sequence is to reset (to zero by the first construction) all the variables that are reset by one of the e_i's. Thus, this sequence can be replaced by a single edge $e = (\ell, g, r, \ell')$ with the same effect, that is, where ℓ is the origin of e_1, ℓ' is the destination of e_n, r resets to zero all the variables that are reset by one of the e_i's and g summarizes the guards of all the e_i's (taking the resets into account). Moreover, we only need to consider sequences where $e_1, e_2, \ldots, e_n$ is a path where each simple loop appears at most once (traversing a second time a simple loop would only reset variables that are already equal to zero because the time delays are null). Thus, the construction amounts to enumerate all the possible paths π where each simple loop appears at most once, and to add to the automaton an edge e_π that summarizes π as described above. Since there are finitely many such π the construction is effective.

As a consequence, to prove decidability of time-bounded reachability of RHA with non-negative rates, we only need to prove that we can decide whether an RHA respecting H_1 through H_3 admits a *strict* run ρ reaching the goal within $\mathbf{T}$ time units, and where all variables are bounded by 1 along ρ.

Bounding the number of equalities. As a first step to obtain a witness of time-bounded reachability, we bound the number of transitions guarded by equalities along a run of bounded duration:

Proposition 2. *Let $\mathcal{H}$ be an LHA, with set of variables X and respecting hypothesis H_1 through H_3. Let ρ be a $\mathbf{T}$-time bounded run of $\mathcal{H}$. Then, ρ contains at most $|X| \cdot \mathrm{rmax} \cdot \mathbf{T}$ transitions guarded by an equality.*

Bounding runs without equalities. Unfortunately, it is not possible to bound the number of transitions that do not contain equalities, even along a time-bounded run. However, we will show that, given a time-bounded run ρ without equality guards, we can build a

run ρ' that is equivalent to ρ (in a sense that its initial and target states are the same), and whose length is *bounded* by a parameter depending on the size of the automaton. More precisely:

Proposition 3. *Let $\mathcal{H}$ be an RHA with non-negative rates. For any* 1*-variable bounded and $\frac{1}{\mathrm{rmax}+1}$-time bounded run $\rho = s_0 \xrightarrow{\pi} s$ of $\mathcal{H}$ that contains no equalities in the guards, $\mathcal{H}$ admits a* 1*-variable bounded and $\frac{1}{\mathrm{rmax}+1}$-time bounded run $\rho' = s_0 \xrightarrow{\pi'} s$ such that $|\rho'| \leq 2|X| + (2|X|+1) \cdot |\mathrm{Loc}| \cdot (2^{(|\mathrm{Edges}|+1)} + 1)$.*

Note that Proposition 3 applies only to runs of duration at most $\frac{1}{\mathrm{rmax}+1}$. However, this is not restrictive, since any $\mathbf{T}$-time-bounded run can always be split into at most $\mathbf{T} \cdot (\mathrm{rmax}+1)$ subruns of duration at most $\frac{1}{\mathrm{rmax}+1}$, provided that we add a self-loop with guard $\mathbf{true}$ and no reset on every location (this can be done without loss of generality as far as reachability is concerned).

To prove Proposition 3, we rely on a *contraction operation* that receives a *timed path* and returns another one of smaller length. Let $\pi = (t_1, e_1), (t_2, e_2), \ldots, (t_n, e_n)$ be a timed path. We define $\mathsf{Cnt}(\pi)$ by considering two cases. Let j, k, j', k' be four positions such that $1 \leq j \leq k < j' \leq k' \leq n$ and $e_j \ldots e_k = e'_j \ldots e'_k$ is a *simple cycle*. **If** such j, k, j', k' exist, then let:

$$\mathsf{Cnt}(\pi) = \pi[1 : j-1] \cdot (e_j, t_j + t_{j'}) \cdots (e_k, t_k + t_{k'}) \cdot \pi[k+1 : j'-1] \cdot \pi[k'+1 : n]$$

Otherwise, we let $\mathsf{Cnt}(\pi) = \pi$. Observe that π and $\mathsf{Cnt}(\pi)$ share the same source and target locations, even when $\pi[k'+1 : n]$ is empty.

Then, given a timed path π, we let $\mathsf{Cnt}^0(\pi) = \pi$, $\mathsf{Cnt}^i(\pi) = \mathsf{Cnt}\left(\mathsf{Cnt}^{i-1}(\pi)\right)$ for any $i \geq 1$, and $\mathsf{Cnt}^*(\pi) = \mathsf{Cnt}^n(\pi)$ where n is the least value such that $\mathsf{Cnt}^n(\pi) = \mathsf{Cnt}^{n+1}(\pi)$. Clearly, since π is finite, and since $|\mathsf{Cnt}(\pi)| < |\pi|$ or $\mathsf{Cnt}(\pi) = \pi$ for any π, $\mathsf{Cnt}^*(\pi)$ always exists. Moreover, we can always bound the length of $\mathsf{Cnt}^*(\pi)$. This stems from the fact that $\mathsf{Cnt}^*(\pi)$ is a timed path that contains at most one occurrence of each simple cycle. The length of such paths can be bounded using classical combinatorial arguments.

Lemma 1. *For any timed path π of an LHA $\mathcal{H}$ with* $|\mathrm{Loc}|$ *locations and* $|\mathrm{Edges}|$ *edges:* $|\mathsf{Cnt}^*(\pi)| \leq |\mathrm{Loc}| \cdot (2^{(|\mathrm{Edges}|+1)} + 1)$.

Note that the contraction operation is purely syntactic and works on the timed path only. Hence, given a run $s_0 \xrightarrow{\pi} s$, we have no guarantee that $\mathsf{Run}(s_0, \mathsf{Cnt}^*(\pi)) \neq \bot$. Moreover, even in the alternative, the resulting run might be $s_0 \xrightarrow{\mathsf{Cnt}^*(\pi)} s'$ with $s \neq s'$. Nevertheless, we can show that $\mathsf{Cnt}^*(\pi)$ preserves some properties of π. For a timed path $\pi = (t_1, e_1), \ldots, (t_n, e_n)$ of an LHA $\mathcal{H}$ with rate function Rates, we let $\mathsf{Effect}(\pi, x) = \sum_{i=1}^{n} \mathrm{Rates}(\ell_i)(x) \cdot t_i$, where ℓ_i is the initial location of e_i for any $1 \leq i \leq n$. Note thus that, for any run $(\ell, \nu) \xrightarrow{\pi} (\ell', \nu')$, for any variable x *which is not reset along* π, $\nu'(x) = \nu(x) + \mathsf{Effect}(\pi, x)$. It is easy to see that $\mathsf{Cnt}^*(\pi)$ preserves the effect of π. Moreover, the duration of $\mathsf{Cnt}^*(\pi)$ and π are equal.

Lemma 2. *For any timed path π: (i)* $\mathsf{duration}(\pi) = \mathsf{duration}(\mathsf{Cnt}^*(\pi))$ *and (ii) for any variable x:* $\mathsf{Effect}(\pi, x) = \mathsf{Effect}(\mathsf{Cnt}^*(\pi), x)$.

We are now ready to show, given a timed path π (with $\mathsf{duration}(\pi) \leq \frac{1}{\mathrm{rmax}+1}$ and without equality tests in the guards), how to build a timed path $\mathsf{Contraction}(\pi)$ that fully preserves the values of the variable, as stated in Proposition 3. The key ingredient to obtain $\mathsf{Contraction}(\pi)$ is to apply Cnt^* to selected portions of π, in such a way that for each edge e that resets a variable for the *first* or the *last* time along π, the time distance between the occurrence of e and the beginning of the timed path is the same in both π and $\mathsf{Contraction}(\pi)$.

The precise construction goes as follows. Let $\pi = (t_1, e_1), \ldots, (t_n, e_n)$ be a timed path. For each variable x, we denote by S_x^π the set of positions i such that e_i is either the *first* or the *last* edge in π to reset x (hence $|S_x^\pi| \in \{0, 1, 2\}$ for any x). Then, we decompose π as: $\pi_1 \cdot (t_{i_1}, e_{i_1}) \cdot \pi_2 \cdot (t_{i_2}, e_{i_2}) \cdots (t_{i_k}, e_{i_k}) \cdot \pi_{k+1}$ with $\{i_1, \ldots, i_k\} = \cup_x S_x^\pi$. From this decomposition of π, we let $\mathsf{Contraction}(\pi) = \mathsf{Cnt}^*(\pi_1) \cdot (t_{i_1}, e_{i_1}) \cdot \mathsf{Cnt}^*(\pi_2) \cdot (t_{i_2}, e_{i_2}) \cdots (t_{i_k}, e_{i_k}) \cdot \mathsf{Cnt}^*(\pi_{k+1})$.

We first note that, thanks to Lemma 1, $|\mathsf{Contraction}(\pi)|$ is bounded.

Lemma 3. *Let $\mathcal{H}$ be an LHA with set of variable X, set of edges* Edges *and set of location* Loc*, and let π be a timed path of $\mathcal{H}$. Then* $|\mathsf{Contraction}(\pi)| \leq 2 \cdot |X| + (2 \cdot |X| + 1) \cdot |\mathrm{Loc}| \cdot (2^{(|\mathrm{Edges}|+1)} + 1)$.

We can now prove Proposition 3.

Proof (Sketch – Proposition 3). Let $\pi = \mathsf{TPath}(\rho)$ and let π' denote $\mathsf{Contraction}(\pi)$. We let $\rho' = s_0 \xrightarrow{\pi'} (\ell', \nu')$, and prove (i) that firing π' from s_0 will always keep all the variable values ≤ 1, which implies $\mathsf{Run}(s_0, \pi') \neq \bot$, and (ii) that $\rho = s_0 \xrightarrow{\pi} (\ell, \nu)$ implies $\ell' = \ell$ and $\nu = \nu'$. These two points hold because $\mathsf{duration}(\mathsf{Cnt}^*(\pi_j)) = \mathsf{duration}(\pi_j)$ for any j. Hence, the first and last resets of each variable happen at the same time (relatively to the beginning of the timed path) in both π and $\mathsf{Contraction}(\pi)$. Intuitively, preserving the time of occurrence of the first reset (of some variable x) guarantees that x will never exceed 1 along $\mathsf{Contraction}(\pi)$, as $\mathsf{duration}(\mathsf{Contraction}(\pi)) = \mathsf{duration}(\pi) \leq \frac{1}{\mathrm{rmax}+1}$. Symmetrically, preserving the last reset of some variable x guarantees that the final value of x will be the same in both π and $\mathsf{Contraction}(\pi)$. Moreover, the contraction preserves the value of the variables that are not reset, by Lemma 2.

Handling '$<$' and non-singular rates. Let us now briefly explain how we can adapt the construction of this section to cope with strict guards and non-singular rates. First, when the RHA $\mathcal{H}$ contains strict guards, the RHA $\mathcal{H}'$ of Proposition 1 will also contain guards with atoms of the form $x < 1$. Thus, when building a 'contracted path' ρ' starting from a path ρ (as in the proof of Proposition 3), we need to ensure that these strict guards will also be satisfied along ρ'. It is easy to use similar arguments to establish this: if some guard $x < 1$ is not satisfied in ρ', this is necessarily before the first reset of x, which means that the guard was not satisfied in ρ either. On the other hand, to take non-singular rates into account, we need to adapt the definition of timed path. A timed path is now of the form $(t_0, r_0, e_0) \cdots (t_n, r_n, e_n)$, where each r_i is a vector of reals of size $|X|$, indicating the actual rate that was chosen for each variable when the i-th continuous step has been taken. It is then straightforward to adapt the definitions of Cnt,

Effect and Contraction to take those rates into account and still keep the properties stated in Lemma 1 and 3 and in Proposition 3 (note that we need to rely on the convexity of the invariants in RHA to ensure that proper rates can be found when building $\mathsf{Cnt}(\pi)$).

Theorem 1. *The time-bounded reachability problem is decidable for the class of rectangular hybrid automata with non-negative rates.*

Proof (Sketch). Given an RHA H, a bound K, and a goal Goal, we can build a formula φ of $\mathsf{FO}(\mathbb{R}, \leq, +)$ that is satisfiable iff $\mathcal{H}$ admits a run of length $\leq K$ reaching Goal. By Proposition 1 (and taking into account the above remarks to cope with strict guards and rectangular rates), this is sufficient to decide time-bounded reachability on RHA with non-negative rates. The required result now follows from the decidability of satisfiability for $\mathsf{FO}(\mathbb{R}, \leq, +)$. □

4 Undecidability Results

In this section, we show that the time-bounded reachability problem for linear hybrid automata becomes undecidable if either both positive and negative rates are allowed, or diagonal constraints are allowed in the guards. Along with the decidability result of Section 3, these facts imply that the class of rectangular hybrid automata having positive rates only and no diagonal constraints forms a maximal decidable class. Our proofs rely on reductions from the halting problem for Minsky two-counters machines.

A *two-counter machine* M consists of a finite set of control states Q, an initial state $q_I \in Q$, a final state $q_F \in Q$, a set C of counters ($|C| = 2$) and a finite set δ_M of instructions manipulating two integer-valued counters. Instructions are of the form:

$q:\ c := c + 1$ **goto** q', or
$q:$ **if** $c = 0$ **then goto** q' **else** $c := c - 1$ **goto** q''.

Formally, instructions are tuples (q, α, c, q') where $q, q' \in Q$ are source and target states respectively, the action $\alpha \in \{inc, dec, 0?\}$ applies to the counter $c \in C$.

A *configuration* of M is a pair (q, v) where $q \in Q$ and $v : C \to \mathbb{N}$ is a valuation of the counters. An *accepting run* of M is a finite sequence $\pi = (q_0, v_0)\delta_0(q_1, v_1)\delta_1 \ldots \delta_{n-1}(q_n, v_n)$ where $\delta_i = (q_i, \alpha_i, c_i, q_{i+1}) \in \delta_M$ are instructions and (q_i, v_i) are configurations of M such that $q_0 = q_I$, $v_0(c) = 0$ for all $c \in C$, $q_n = q_F$, and for all $0 \leq i < n$, we have $v_{i+1}(c) = v_i(c)$ for $c \neq c_i$, and (i) if $\alpha = inc$, then $v_{i+1}(c_i) = v_i(c_i) + 1$, (ii) if $\alpha = dec$, then $v_i(c_i) \neq 0$ and $v_{i+1}(c_i) = v_i(c_i) - 1$, and (iii) if $\alpha = 0?$, then $v_{i+1}(c_i) = v_i(c_i) = 0$. The *halting problem* asks, given a two-counter machine M, whether M has an accepting run. This problem is undecidable [9].

Undecidability for RHA with negative rates. Given a two-counter machine M, we construct an RHA $\mathcal{H}_M$ (thus without diagonal constraints) such that M has an accepting run if and only if the answer to the time-bounded reachability problem for $(\mathcal{H}_M, \mathrm{Goal})$ with time bound 1 is YES. The construction of $\mathcal{H}_M$ crucially makes use of both positive and negative rates.

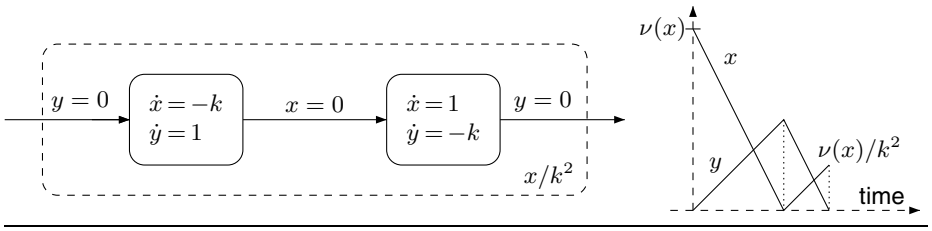


Fig. 1. Gadget for division of a variable x by k^2. The variable y is internal to the gadget. The duration of the division is $v \cdot (\frac{1}{k} + \frac{1}{k^2})$ where v is the value of x before division.

Theorem 2. *The time-bounded reachability problem is undecidable for rectangular hybrid automata even if restricted to singular rates.*

Proof (Sketch). First, remark that the main difficulty of the reduction is to encode *unbounded* computations of M within a *bounded* time slot. The execution steps of M are simulated in $\mathcal{H}_M$ by a (possibly infinite) sequence of *ticks* within one time unit. The ticks occur at time $t_0 = 0, t_1 = 1 - \frac{1}{4}, t_2 = 1 - \frac{1}{16}$, etc. The counters are encoded as follows. If the value of counter $c \in C$ after i execution steps of M is $v(c)$, then the variable x_c in $\mathcal{H}_M$ has value $\frac{1}{4^{i+v(c)}}$ at time t_i. Note that this encoding is time-dependent and that the value of x_c at time t_i is always smaller than $1 - t_i = \frac{1}{4^i}$, and equal to $\frac{1}{4^i}$ if the counter value is 0. To maintain this encoding (if a counter c is not modified in an execution step), we need to divide x_c by 4 before the next tick occurs. We use the divisor gadget in Fig. 1 to do this. Using the diagram in the figure, it is easy to check that the value of variable x_c is divided by k^2 where k is a constant used to define the variable rates. Note also that the division of $\nu(x_c)$ by k^2 takes $\nu(x_c) \cdot (\frac{1}{k} + \frac{1}{k^2})$ time units, which is less than $\frac{3 \cdot \nu(x_c)}{4}$ for $k \geq 2$. Since $\nu(x_c) \leq \frac{1}{4^i}$ at step t_i, the duration of the division is at most $\frac{3}{4^i} = t_{i+1} - t_i$, the duration of the next tick.

The divisor gadget can also be used to construct an automaton $\mathcal{A}_{\mathsf{tick}}$ that generates the ticks. Finally, we obtain $\mathcal{H}_M$ by taking the product of $\mathcal{A}_{\mathsf{tick}}$ with an automaton that encodes the instructions of the machine. For example, assuming the set of counters is $C = \{c, d\}$ the instruction (q, inc, c, q') is encoded by connecting a location ℓ_q to a location $\ell_{q'}$, synchronized with divisor gadgets that divide x_c by 16 and x_d by 4 (details omitted).

Undecidability with diagonal constraints. We now show that diagonal constraints also leads to undecidability. The result holds even if every variable has a positive, singular, fixed rate.

Theorem 3. *The time-bounded reachability problem is undecidable for LHA that use only singular, strictly positive, and fixed-rate variables.*

Proof. The proof is again by reduction from the halting problem for two-counter machines. We describe the encoding of the counters and the simulation of the instructions.

Given a counter c, we represent c via two auxiliary counters c_{bot} and c_{top} such that $v(c) = v(c_{\mathrm{top}}) - v(c_{\mathrm{bot}})$.

Incrementing and decrementing c are achieved by incrementing either $c_{\rm top}$ or $c_{\rm bot}$. Zero-testing for c corresponds to checking whether the two auxiliary counters have the same value. Therefore, we do not need to simulate decrementation of a counter.

We encode the value of counter $c_{\rm bot}$ using two real-valued variables x and y, by postulating that $|x-y| = \frac{1}{2^{v(c_{\rm bot})}}$. Both x and y have rate $\dot{x} = \dot{y} = 1$ at all times and in all locations of the hybrid automaton. Incrementing $c_{\rm bot}$ now simply corresponds to halving the value of $|x-y|$. In order to achieve this, we use two real-valued variables z and w with rate $\dot{z} = 2$ and $\dot{w} = 3$.

All operations are simulated in 'rounds'. At the beginning of a round, we require that the variables x, y, z, w have respective value $\frac{1}{2^{v(c_{\rm bot})}}, 0, 0, 0$. We first explain how we merely *maintain* the value of $c_{\rm bot}$ throughout a round:

1. Starting from the beginning of the round, let all variables evolve until $x = z$, which we detect via a diagonal constraint. Recall that z evolves at twice the rate of x.
2. At that point, $x = \frac{2}{2^{v(c_{\rm bot})}}$ and $y = \frac{1}{2^{v(c_{\rm bot})}}$. Reset x and z to zero.
3. Now let all variables evolve until $y = z$, and reset y, z and w to zero. It is easy to see that all variables now have exactly the same values as they had at the beginning of the round. Moreover, the invariant $|x-y| = \frac{1}{2^{v(c_{\rm bot})}}$ is maintained throughout.

Note that the total duration of the above round is $\frac{2}{2^{v(c_{\rm bot})}}$. To *increment* $c_{\rm bot}$, we proceed as follows:

1′. Starting from the beginning of the round, let all variables evolve until $x = w$. Recall that the rate of w is three times that of x.
2′. At that point, $x = \frac{1.5}{2^{v(c_{\rm bot})}}$ and $y = \frac{0.5}{2^{v(c_{\rm bot})}} = \frac{1}{2^{v(c_{\rm bot})+1}}$. Reset x, z, and w to zero.
3′. Now let all variables evolve until $y = z$, and reset y, z and w to zero. We now have $x = \frac{1}{2^{v(c_{\rm bot})+1}}$, and thus the value of $|x-y|$ has indeed been halved as required.

Note that the total duration of this incrementation round is $\frac{1}{2^{v(c_{\rm bot})}}$, where $v(c_{\rm bot})$ denotes the value of counter $c_{\rm bot}$ prior to incrementation.

Clearly, the same operations can be simulated for counter $c_{\rm top}$ (using further auxiliary real-valued variables). Note that the durations of the rounds for $c_{\rm bot}$ and $c_{\rm top}$ are in general different—in fact $c_{\rm bot}$-rounds are never faster than $c_{\rm top}$-rounds. But because they are powers of $\frac{1}{2}$, it is always possible to synchronize them, simply by repeating maintain-rounds for $c_{\rm bot}$ until the round for $c_{\rm top}$ has completed.

Finally, zero-testing the original counter c (which corresponds to checking whether $c_{\rm bot} = c_{\rm top}$) is achieved by checking whether the corresponding variables have the same value at the very beginning of a $c_{\rm bot}$-round (since the $c_{\rm bot}$- and $c_{\rm top}$-rounds are then synchronized).

We simulate the second counter d of the machine using further auxiliary counters $d_{\rm bot}$ and $d_{\rm top}$. It is clear that the time required to simulate one instruction of a two-counter machine is exactly the duration of the slowest round. Note however that since counters $c_{\rm bot}$, $c_{\rm top}$, $d_{\rm bot}$, and $d_{\rm top}$ are never decremented, the duration of the slowest round is at most $\frac{2}{2^p}$, where p is the smallest of the initial values of $c_{\rm bot}$ and $d_{\rm bot}$. If a two-counter machine has an accepting run of length m, then the total duration of the simulation is at most $\frac{2m}{2^p}$.

In order to bound this value, it is necessary before commencing the simulation to initialize the counters c_{bot}, c_{top}, d_{bot}, and d_{top} to a sufficiently large value, for example any number greater than $\log_2(m) + 1$. In this way, the duration of the simulation is at most 1.

Initializing the counters in this way is straightforward. Starting with zero counters (all relevant variables are zero) we repeatedly increment c_{bot}, c_{top}, d_{bot}, and d_{top} a nondeterministic number of times, via a self-loop. When each of these counters has value k, we can increment all four counters in a single round of duration $\frac{1}{2^k}$ as explained above. So over a time period of duration at most $\sum_{k=0}^{\infty} \frac{1}{2^k} = 2$ the counters can be initialized to $\lceil \log_2(m) + 1 \rceil$.

Let us now combine these ingredients. Given a two-counter machine M, we construct a hybrid automaton $\mathcal{H}_M$ such that M has an accepting run iff $\mathcal{H}_M$ has a run of duration at most 3 that reaches the final state Goal.

$\mathcal{H}_M$ uses the real-valued variables described above to encode the counters of M. In the initialization phase, $\mathcal{H}_M$ nondeterministically assigns values to the auxiliary counters, hence guessing the length of an accepting run of M, and then proceeds with the simulation of M. This ensures a correspondence between an accepting run of M and a time-bounded run of $\mathcal{H}_M$ that reaches Goal.

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. TCS 138(1) (1995)
2. Alur, R., Dill, D.L.: A theory of timed automata. Th. Comp. Sci. 126(2), 183–235 (1994)
3. Cassez, F., Larsen, K.G.: The impressive power of stopwatches. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 138–152. Springer, Heidelberg (2000)
4. Ferrante, J., Rackoff, C.: A decision procedure for the first order theory of real addition with order. SIAM J. Comput. 4(1), 69–76 (1975)
5. Frehse, G.: Phaver: algorithmic verification of hybrid systems past hytech. Int. J. Softw. Tools Technol. Transf. 10, 263–279 (2008)
6. Henzinger, T.A., Ho, P.-H., Wong-Toi, H.: Hytech: A model checker for hybrid systems. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 460–463. Springer, Heidelberg (1997)
7. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? J. Comput. Syst. Sci. 57(1), 94–124 (1998)
8. Henzinger, T.A., Raskin, J.-F.: Robust undecidability of timed and hybrid systems. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC 2000. LNCS, vol. 1790, pp. 145–159. Springer, Heidelberg (2000)
9. Minsky, M.L.: Computation: finite and infinite machines. Prentice-Hall Series in Automatic Computation. Prentice-Hall Inc., Englewood Cliffs (1967)
10. Ouaknine, J., Rabinovich, A., Worrell, J.: Time-bounded verification. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 496–510. Springer, Heidelberg (2009)
11. Ouaknine, J., Worrell, J.: Towards a theory of time-bounded verification. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 22–37. Springer, Heidelberg (2010)

Deciding Robustness against Total Store Ordering

Ahmed Bouajjani[1], Roland Meyer[2], and Eike Möhlmann[3]

[1] LIAFA, University Paris 7
abou@liafa.jussieu.fr
[2] University of Kaiserslautern
meyer@cs.uni-kl.de
[3] University of Oldenburg
eike.moehlmann@informatik.uni-oldenburg.de

Abstract. We address the problem of deciding robustness of a program against the total store ordering (TSO) relaxed memory model, i.e., of checking whether the behaviour under TSO coincides with the expected sequential consistency (SC) semantics. We prove that this problem is PSPACE-complete. The key insight is that violations to robustness can be detected on pairs of SC computations.

1 Introduction

Sequential consistency (SC) is the classical memory model for concurrent programs with a shared memory [Lam79]. It interleaves the parallel computations but keeps the order of actions issued by a same component. The model is intuitive and programmers usually assume the execution environment adheres to it. However, to reduce the latency of memory accesses, modern processors and compilers adopt relaxed memory models that can reorder certain accesses [AG96]. Due to the unexpected behaviours they may introduce, program order relaxations make reasoning about concurrent programs highly complex. For instance, known concurrent algorithms like Dekker's protocol for mutual exclusion turn out incorrect in presence of the write-to-read relaxations discussed in this paper. Indeed, data race freedom (DRF) guarantees sequential consistency. But the DRF assumption is not valid for many performance-critical services such as implementations of synchronisation operations, transactional memories, or concurrency libraries.

Therefore, an important problem is to check that a program $\mathcal{P}$ is robust against a relaxed memory model M: running $\mathcal{P}$ under M does not introduce computations that are impossible under SC. More precisely, $\mathcal{P}$ is robust against M if for every sequence of actions σ in $\mathcal{P}$ the following holds. If there is a computation τ under M that corresponds to σ, then σ is necessarily executable under SC.

We settle decidability and complexity of the robustness problem against the total store ordering (TSO) memory model [OSS09, WG94]. TSO applies write-to-read relaxations that let reads overtake earlier write actions, a core feature in many common relaxed memory models [BM08]. The operational intuition is that of write buffers. TSO architectures provide FIFO buffers that hold the write actions for later execution. A read, say on variable x, then fetches the last write to x from the buffer. If there is no such write, the read consults the memory, thereby overtaking the pending writes in the

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 428–440, 2011.

buffer. Although write buffers are necessarily finite in actual machines, no fixed bound on their size must be assumed to reason about the correctness of general algorithms.

Deciding robustness against TSO is non-trivial due to the unboundedness of the FIFO buffers and, as far as we know, has not been addressed. We prove the problem decidable and PSPACE-complete. Our result is based on a deep investigation of the properties of TSO computations. We establish the surprising fact that TSO violations to robustness can already be found in the SC computations: there is a TSO computation without an SC counterpart if and only if there is a *pair of conflicting SC computations*. Roughly, these computations employ cyclic data races. Bounds on their sizes then yield a decision procedure that requires space linear in the number of states and the number of variables. PSPACE-hardness follows from the hardness of SC reachability [SC85].

Technically, we take a language-theoretic view that captures TSO by a closure of the computations under the reorderings explained above. The tool that facilititates the proofs are *minimal violating computations* with the least number of reorderings. From this minimality, we first deduce that only a single program has performed reorderings. In this sense, minimal violating computations apply reorderings locally. We then draw conclusions about the shape of the TSO computation in order to construct the pair of conflicting SC computations. The arguments have some combinatorial flavour.

Related work. A computation is not SC iff it accesses variables in a cyclic manner [SS88]. Shasha and Snir proved this result without reference to a specific environment. We adopt their formalism and extend the study in an algorithmic direction. For TSO, existence of computations with cyclic dependencies is decidable.

There are algorithms and tools for monitoring TSO computations [BM08, BSS11]. They detect non-robustness by enumerating bounded executions, but do not provide a decision procedure. Beyond verifications, algorithms exist that insert commands to ensure robustness of a program against some memory model. While [SS88] use a static analysis, [AMSS10, Alg10, BAM07] rely on testing. A state based variant of robustness is considered in [KVY10]. These authors achieve decidability by bounding the buffers. As the complexity depends on this bound, our decision procedure for unbounded buffers improves even over this finite state approach.

Owens considers a notion of robustness based on memory equivalence, and gives a characterisation of non-robust programs via triangular data races in SC [Owe10]. The trace equivalence [SS88] considered here is more liberal than memory equivalence, with the advantage of tolerating harmless triangular data races and thus permitting additional relaxed executions. Therefore, our main result strengthens triangular data races to pairs of data races in two SC computations. We settle the complexity of finding such pairs.

There are only few results on decidability and complexity of relaxed memory models. Recently, reachability under TSO has been shown to be decidable but non-primitive recursive [ABBM10]. This is in sharp contrast to our result. Being PSPACE-complete, robustness turns out more tractable. Alur et al. showed model checking of an implementation wrt. an SC specification undecidable [AMP96]. This does not contradict our findings since we fix the implementation to be the TSO semantics of the specification. In [GK97], the problem of deciding whether a given computation is SC feasible has been proved NP-complete. Robustness is concerned with all TSO computations, instead.

Structure of the paper. We introduce the formal framework in Section 2. Section 3 is devoted to minimal violating computations and their reduction to pairs of conflicting SC computations. This is our main result. In Section 4, we asses the complexity of the resulting decision procedure for robustness, before concluding in Section 5.

2 Programs, Memory Models, and Robustness

2.1 Concurrent Programs

We study concurrent programs that operate on a shared memory. A *concurrent program* $\mathcal{P}$ consists of several *component programs* $\mathcal{P} = \{p_1, \ldots, p_n\}$. The shared memory is modelled by a set of *variables* $\mathcal{V}$ that take values from the Boolean domain $\mathcal{B} := \{1,0\}$. The component programs either *write* to (w) or *read* from (r) variables. The overall set of *commands* is $\mathcal{C} := \{w, r, f\}$, where the additional *fence* instruction (f) restricts the behaviour of the TSO model. We discuss its semantics below.

Component programs access variables with *actions* in $\mathcal{A} := \mathcal{C} \times \mathcal{V} \times \mathcal{B} \times \mathcal{P}$. For example, using action $a = (w, x, 1, p)$ component p sets variable x to 1. Since variable and value are unimportant for fence actions, we denote them (f, p). By $com(a) := w$, $var(a) := x$, $val(a) := 1$, and $prog(a) := p$ we extract the *command*, the *variable*, the *value*, and the *program* in action a. We use a, b, c for actions, x, y, z for variables, and v for an arbitrary value. We overload w and r to also stand for write and read actions.

The components $p \in \mathcal{P}$ are represented by finite automata $(Q, q_0, \rightarrow)$ with *states* Q, *initial state* q_0, and *transition relation* $\rightarrow \subseteq Q \times \mathcal{A} \times Q$. We typically write $q_1 \xrightarrow{a} q_2$ instead of $(q_1, a, q_2) \in \rightarrow$ and expect program p to have actions in $\mathcal{C} \times \mathcal{V} \times \mathcal{B} \times \{p\}$.

Example 1 (Dekker). Under SC, the following algorithm guarantees mutual exclusion between two components. By setting a variable x to 1, action w_x, the first program signals its wish to enter the critical section. If it then finds variable y of the partner 0, read r_y, it enters. With $\mathbf{w_y}$ and $\mathbf{r_x}$, the second program behaves similarly. We insert further actions to explain our formalism. The resulting program is $\mathcal{D} = \{p_1, p_2\}$ where

$$p_1 : q_0 \xrightarrow{w_x} q_1 \xrightarrow{w_z} q_2 \xrightarrow{r_z} q_3 \xrightarrow{r_y} cs \qquad p_2 : \tilde{q}_0 \xrightarrow{\mathbf{w_z}} \tilde{q}_1 \xrightarrow{\mathbf{w_y}} \tilde{q}_2 \xrightarrow{\mathbf{f}} \tilde{q}_3 \xrightarrow{\mathbf{r_x}} \tilde{cs}$$

$$w_x := (w, x, 1, p_1) \quad r_z := (r, z, 1, p_1) \qquad \mathbf{w_z} := (w, z, 0, p_2) \quad \mathbf{f} := (f, p_2)$$

$$w_z := (w, z, 1, p_1) \quad r_y := (r, y, 0, p_1) \qquad \mathbf{w_y} := (w, y, 1, p_2) \quad \mathbf{r_x} := (r, x, 0, p_2).$$

As we shall see, mutual exclusion fails under TSO.

2.2 Sequential Consistency and Total Store Ordering Semantics

In a *sequentially consistent (SC)* environment, the component programs directly write to the memory. So the ordering of actions issued by one component is preserved, but the computations of several components may be interleaved. With this, $\sigma = w_x.\mathbf{w_z}.w_z.r_z.\mathbf{w_y}$ is a feasible SC computation of Dekker's algorithm but $\tau = r_y.\mathbf{w_z}.\mathbf{w_y}.\mathbf{f}.\mathbf{r_x}.w_x.w_z$ is not.

Total store ordering (TSO) memories provide, for each component, a FIFO buffer that holds the write actions, Figure 1. A read (r, x, v, p) then *prefetches* the last write to x from the buffer of p. If no such write is found, the read receives the value from the

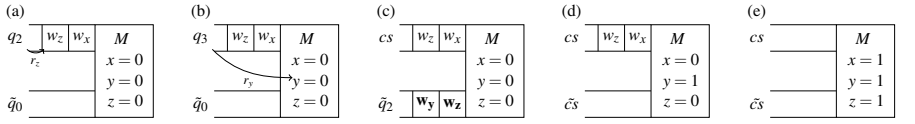


Fig. 1. Feasible computation of the algorithm in Example 1 on a TSO memory. Program p_1 adds writes w_x and w_z to its buffer. Then it prefetches r_z (a). When p_1 executes the read on y, there is no corresponding write in its buffer: r_y is the first command to interact with the memory (b). Afterwards p_2 issues its commands where the fence forbids overtakes of r_x (c) and (d). Only as a last step, the memory executes the writes to x and z from p_1 (e). The resulting feasible TSO computation is $\tau = r_y.\mathbf{w_z}.\mathbf{w_y}.\mathbf{f}.\mathbf{r_x}.w_x.w_z$. Since the memory does not see r_z the read does not occur in τ. Both components have entered their critical sections and mutual exclusion fails.

memory. We say the read has *overtaken* the writes in the buffer: it was issued later than the pending writes but executed earlier. In this sense, our formalism takes a view from the memory. A fence ensures that a read is executed only when the buffer is empty.

To formalise the above behaviour, we extend the transition relation to *computations*, sequences of actions $\sigma = a_1 \dots a_n$ in $\mathcal{A}^*$. Syntax $q \xrightarrow{\sigma}$ means there are states $q_1, \dots, q_n$ so that $q \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_n$. The *language* $\mathcal{L}(p)$ of a component $p \in \mathcal{P}$ contains all computations from the initial state, $\mathcal{L}(p) := \{\sigma \in \mathcal{A}^* \mid q_0 \xrightarrow{\sigma}\}$.

Prefetching and overtaking of reads in TSO is reflected by a *closure operator* on the languages $\mathcal{L}(p)$ of the components. It allows for the following rewritings:

$$(w,x,*,p).(r,y,*,p) \curvearrowright_{re} (r,y,*,p).(w,x,*,p) \quad \text{with } x \neq y, \tag{Reorder}$$

$$(w,x,v,p).(r,x,v,p) \curvearrowright_{pf} (w,x,v,p). \tag{Prefetch}$$

Fence instructions are never reordered and so act blocking. We call *TSO rewriting* the smallest relation $\curvearrowright \subseteq \mathcal{A}^* \times \mathcal{A}^*$ that includes $\curvearrowright_{re}$, $\curvearrowright_{pf}$ and is closed under composition, $\alpha \curvearrowright \beta$ implies $\gamma_1.\alpha.\gamma_2 \curvearrowright \gamma_1.\beta.\gamma_2$. Given $\mathcal{L} \subseteq \mathcal{A}^*$, we denote its *closure under TSO rewriting* by $cl(\mathcal{L}) := \{\tau \in \mathcal{A}^* \mid \sigma \curvearrowright^* \tau \text{ for } \sigma \in \mathcal{L}\}$. We use α, β, γ for computations but reserve τ for *TSO computations* in $cl(\mathcal{L})$ and σ for *SC computations* in $\mathcal{L}$.

Example 2. TSO rewriting turns $w_x.w_z.r_z.r_y \in \mathcal{L}(p_1)$ into $r_y.w_x.w_z \in cl(\mathcal{L}(p_1))$. Note that $cl(\mathcal{L}(p_2)) = \mathcal{L}(p_2)$ due to the fence and the fact that $\mathbf{w_z}$ and $\mathbf{w_y}$ cannot be swapped.

The SC and TSO computations of $\mathcal{P}$ interleave the corresponding computations in the components $p \in \mathcal{P}$. This interleaving is formalised by *shuffling*, defined in the standard way $a.\alpha \sqcup\!\sqcup b.\beta := a.(\alpha \sqcup\!\sqcup b.\beta) \cup b.(a.\alpha \sqcup\!\sqcup \beta)$.

Shuffling may yield computations $(w,y,1,p_2).(r,y,0,p_1)$ that are not feasible on memory. We formalise *memory valuations* as vectors $m \in \mathcal{B}^{\mathcal{V}}$ that assign a value $m(x)$ in $\mathcal{B}$ to every variable. Actions then define a partial transition function $[\rangle \subseteq \mathcal{B}^{\mathcal{V}} \times \mathcal{A} \times \mathcal{B}^{\mathcal{V}}$ between valuations as follows. Writes update their variable, $m[(w,x,v,p)\rangle m\{v/x\}$ with $m\{v/x\}(x) = v$ and $m\{v/x\}(y) = m(y)$ for $y \neq x$. Reads have a blocking behaviour, $m[(r,x,v,p)\rangle m$ only if $m(x) = v$. The function is undefined otherwise. Fences do not influence the computation, $m[(f,p)\rangle m$. We again extend $[\rangle$ to computations in $\mathcal{A}^*$.

The *initial valuation* m_0 sets all variables to zero, $m_0(x) = 0$ for all $x \in \mathcal{V}$. We denote the restriction of $\mathcal{L} \subseteq \mathcal{A}^*$ to the *feasible computations* by $\mathcal{L}_\checkmark := \{\sigma \in \mathcal{L} \mid m_0[\sigma\rangle\}$.

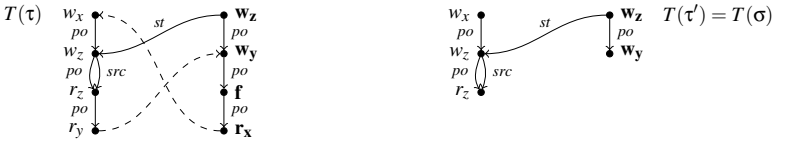


Fig. 2. Traces $T(\tau)$ with $\tau = r_y.\mathbf{w_z}.\mathbf{w_y}.\mathbf{f}.\mathbf{r_x}.w_x.w_z$ and $T(\tau') = T(\sigma)$ with $\tau' = w_x.\mathbf{w_z}.w_z.\mathbf{w_y}$ and $\sigma = w_x.\mathbf{w_z}.w_z.r_z.\mathbf{w_y}$. Dashed lines represent the conflict relations defined in Section 2.4.

Example 3. Computation $\tau = r_y.\mathbf{w_z}.\mathbf{w_y}.\mathbf{f}.\mathbf{r_x}.w_x.w_z$ is in $(cl(\mathcal{L}(p_1)) \sqcup\!\sqcup cl(\mathcal{L}(p_2)))_{\swarrow}$.

2.3 Traces

Shasha and Snir proposed *traces* $T(\sigma)$ and $T(\tau)$ that abstract from computations [SS88]. Figure 2 contains two examples that illustrate the following definition. Traces consist of three relations that are derived from the computation and keep information about the data and control dependencies. First, they represent the ordering of commands issued by one component, known as *program ordering*. Furthermore, they preserve the ordering of write accesses to each variable, the *store ordering*. Finally, traces indicate the write command that a read receives its value from, the *source relation* from writes to reads.

To formalise traces, we need some syntax. Consider computation $\alpha = a_1 \dots a_n$. We refer to its actions by $\mathcal{A}(\alpha) := \{a_1, \dots, a_n\}$, assuming an index that keeps them distinct. We write $\alpha \uparrow p$ for the projection of α to the actions of program $p \in \mathcal{P}$, and $\alpha \uparrow (w,x)$ for the projection of α to the writes to x. Every computation defines an ordering $\leq_\alpha$ on its actions with $a \leq_\alpha b$ if $\alpha = \alpha_1.a.\alpha_2.b.\alpha_3$. We say that a is *earlier* and b is *later* in α. While $\leq_\alpha$ is transitive, the successor relation $\rightarrow_\alpha$ is not. We set $a \rightarrow_\alpha b$ if $\alpha = \alpha_1.a.b.\alpha_2$.

Let $\tau \in (\sqcup\!\sqcup_{p\in\mathcal{P}} \tau_p)_{\swarrow}$ be a feasible TSO computation that is derived from $\sigma_p \curvearrowright^* \tau_p$. The *program order of component p in* τ, denoted by $\rightarrow^p_{po}$, is the ordering the actions of p had in the SC computation σ_p, before rewriting. Formally, we use the successor relation of σ_p and set $\rightarrow^p_{po} := \rightarrow_{\sigma_p}$. The *program order relation of* τ is $\rightarrow_{po} := \bigcup_{p\in\mathcal{P}} \rightarrow^p_{po}$.

The *ordering of stores to variable x*, denoted by $\rightarrow^x_{st}$, gives the ordering of write actions to variable x in τ. More formally, it is the successor relation of $\tau \uparrow (w,x)$. The *ordering of stores in* τ is again the union $\rightarrow_{st} := \bigcup_{x\in\mathcal{V}} \rightarrow^x_{st}$.

The *source relation* $\rightarrow_{src}$ requires a case distinction. If read $b = (r,x,v,p)$ in $\sigma_p = \sigma_1.b.\sigma_2$ is not among the actions of τ then it prefetched its value from the latest write to x in the buffer. We therefore set $a \rightarrow_{src} b$ with a the latest action in $\sigma_1 \uparrow (w,x)$. Read r_z in Figure 2 serves as an example. If $\tau = \tau_1.b.\tau_2$, then b reads the latest write a to x in τ_1 and we again set $a \rightarrow_{src} b$. If there is no such write, the read receives the initial value.

With these relations, the *trace of* τ is $T(\tau) := (\bigcup_{p\in\mathcal{P}} \mathcal{A}(\sigma_p), \rightarrow_{po}, \rightarrow_{st}, \rightarrow_{src})$. We denote by $\mathcal{T}_{TSO}(\mathcal{P})$ the set of all *TSO traces of* $\mathcal{P}$, traces that belong to feasible TSO computations. Similarly, $\mathcal{T}_{SC}(\mathcal{P})$ is the subset of all *SC traces* $T(\sigma)$ obtained from SC computations $\sigma \in (\sqcup\!\sqcup_{p\in\mathcal{P}} \sigma_p)_{\swarrow}$. Robustness checks whether the two sets coincide:

Rob Given a program $\mathcal{P}$, the *robustness problem* asks for $\mathcal{T}_{TSO}(\mathcal{P}) \subseteq \mathcal{T}_{SC}(\mathcal{P})$.

We say that τ *violates robustness* if $T(\tau) \in \mathcal{T}_{TSO}(\mathcal{P}) \setminus \mathcal{T}_{SC}(\mathcal{P})$. We also call τ *violating*.

2.4 Happens Before

Shasha and Snir observed that violating computations use cyclic accesses to variables that SC is unable to serialise. To make them visible in traces, they suggested to add a *conflict relation* from reads to writes. Intuitively, $r \rightarrow_{cf} w$ means that w overwrites the value that r intends to read. In Figure 2, $T(\tau)$ has two conflicting accesses. Lemma 1 shows that the trace is violating as the conflicts close a cycle.

Consider $\tau \in (\sqcup\!\sqcup_{p\in\mathcal{P}}\tau_p)_{\checkmark}$ with $\sigma_p \curvearrowright^* \tau_p$. Technically, its *conflict relation* $\rightarrow_{cf}$ is derived from the store ordering $\rightarrow_{st}$ and source relation $\rightarrow_{src}$. We have $r \rightarrow_{cf} w$ if there is a write action $\tilde{w} \in \mathcal{A}(\tau)$ with $\tilde{w} \rightarrow_{src} r$ and $\tilde{w} \rightarrow_{st} w$. If r reads the initial value of a variable and w overwrites it, i.e., $var(r) = var(w)$ and there is no $w_1 \rightarrow_{src} r$ and no $w_2 \rightarrow_{st} w$, we also have $r \rightarrow_{cf} w$. Figure 2 illustrates the situation on x and y. The union of all relations we defined is commonly called *happens before relation* of the computation $\rightarrow_{hb} := \rightarrow_{po} \cup \rightarrow_{st} \cup \rightarrow_{src} \cup \rightarrow_{cf}$.

Lemma 1 ([SS88]). *Trace $T(\tau)$ is in $\mathcal{T}_{SC}(\mathcal{P})$ if and only if $\rightarrow_{hb}$ is acyclic.*

We often need information about the actions on a happens before path. Let $\tau = \alpha.a.\beta.b.\gamma$. By definition, $c_1 \ldots c_n$ is a *subword* of β if $\beta = \beta_1.c_1\beta_2 \ldots \beta_n.c_n.\beta_{n+1}$. We say a happens before b *through* β if there is a subword $c_1 \ldots c_n$ of β that yields a happens before path from a to b. More precisely, with $c_0 := a$ and $c_{n+1} := b$ we require that either $c_i \rightarrow_{hb} c_{i+1}$ directly or $c_i \rightarrow^+_{po} c_{i+1}$ for all $0 \leq i \leq n$. Note that a happens before b through β holds as soon as both actions access the same variable (no fences) and one is a write.

Lemma 2. *Consider a computation $\alpha.a.\beta.b.\gamma \in (\sqcup\!\sqcup_{p\in\mathcal{P}}\tau_p)_{\checkmark}$. If $var(a) = var(b)$ and $com(a) \neq f \neq com(b)$ and $com(a) = w$ or $com(b) = w$ then $a \rightarrow^+_{hb} b$ through β.*

As a consequence, if $a \rightarrow^+_{hb} b$ through a subword of β then there is a happens before path through β as well.

Lemma 3. *Consider $\tau_1 = \alpha_1.a.\beta_1.b.\gamma_1$ and $\tau_2 = \alpha_2.a.\beta_2.b.\gamma_2$ with $\tau_1, \tau_2 \in (\sqcup\!\sqcup_{p\in\mathcal{P}}\tau_p)_{\checkmark}$. Let β_1 be a subword of β_2. If $a \rightarrow^+_{hb} b$ through β_1 then $a \rightarrow^+_{hb} b$ through β_2.*

3 SC Characterisation of Violating TSO Computations

We develop a characterisation of violating TSO computations, i.e., computations whose traces are not sequentially consistent. Surprisingly, the characterisation is *independent* of the total store ordering model. Instead, it makes use of *pairs of SC computations* that conflict in their accesses to variables.

We obtain the result by a close examination of violating TSO computations. The key idea is to select a violating computation with the least possible number of reorderings. From this minimality, we deduce information about the shape of the computation. The proofs have a combinatorial flavour in that we derive the results by contradiction to the minimality assumption.

The structure of the argumentation is as follows. We formalise minimal violating computations and show that they use TSO rewriting in a restricted way: *only a single component employs reorderings, the remaining programs have SC computations*. Even more, we show that essentially a single write needs to be overtaken to obtain a violation,

Section 3.2. We then split up the violating TSO computation into two SC computations, our main result in Section 3.3. To argue for feasibility of the new computations, we observe that certain parts of the TSO computation access distinct variables.

3.1 Minimal Violating Computations

We focus on TSO computations that have few reorderings (this notion of minimality is not related to the shortest happens before cycles in [SS88]). For a single program $p \in \mathcal{P}$, the *number of reorderings in* $\tau_p \in cl(\mathcal{L}(p))$ is the length of its shortest $\curvearrowright$ derivation:

$$\#(\tau_p) := min\{n \in \mathbb{N} \mid \sigma_p \curvearrowright^n \tau_p \text{ for } \sigma_p \in \mathcal{L}(p)\}.$$

For a computation $\tau \in (\sqcup\!\sqcup_{p\in\mathcal{P}}\tau_p)_{\swarrow}$ of the parallel program, we sum up the values $\#(\tau_p)$ for the components' computations: $\#(\tau) := \Sigma_{p\in\mathcal{P}}\#(\tau_p)$. We call τ a *minimal violating computation* if its number of reorderings is minimal among the violating computations. There is no violation τ' with $\#(\tau') < \#(\tau)$. It is convenient to always keep a *shortest derivation* $\sigma_p \curvearrowright^n \tau_p$ for the computations of the components that satisfies $n = \#(\tau_p)$.

Example 4. We have $r_y.\mathbf{w_z}.\mathbf{w_y}.\mathbf{f}.\mathbf{r_x}.w_x.w_z$ as minimal violation with three reorderings. If the fence in p_2 is omitted, $\mathbf{w_z}.\mathbf{r_x}.w_x.w_z.r_z.r_y.\mathbf{w_y}$ with a single reordering is minimal.

In a minimal violation, every prefetch $(w,x,v,p).(r,x,v,p) \curvearrowright_{pf} (w,x,v,p)$ is justified by a later reordering $(w,x,v,p).(r,y,v',p) \curvearrowright_{re} (r,y,v',p).(w,x,v,p)$. Computation τ' in Figure 2 explains the intuition. Since a later reordering over w_z is missing, the prefetch by r_z can be avoided to reduce $\#(\tau')$. Computation σ shows how to place the read.

Lemma 4. *Consider a minimal violation* $\tau \in (\sqcup\!\sqcup_{p\in\mathcal{P}}\tau_p)_{\swarrow}$. *If the shortest derivation* $\sigma_p \curvearrowright^* \tau_p$ *contains a prefetch* $\alpha.w.r_1.\beta \curvearrowright_{pf} \alpha.w.\beta$, *then there is a read* $r_2 \in \mathcal{A}(\tau_p)$ *so that* $r_1 \leq_{\sigma_p} r_2$ *and* $r_2 \leq_{\tau_p} w$.

Consider a minimal violation $\alpha.r.\beta.w.\gamma$ where read r has overtaken write w. The next lemma provides a happens before path from r to w through β. Otherwise, the reordering could have been avoided without changing the trace — a contradiction to minimality.

Lemma 5. *Consider a minimal violating computation* $\alpha.a.\beta.b.\gamma$ *in* $(\sqcup\!\sqcup_{p\in\mathcal{P}}\tau_p)_{\swarrow}$ *with shortest derivations* $\sigma_p \curvearrowright^* \tau_p$. *Then a* $\rightarrow_{hb}^{+}$ *b through* β *or there is* $\alpha.\beta_1.b.a.\beta_2.\gamma$ *in* $(\sqcup\!\sqcup_{p\in\mathcal{P}}\tau_p)_{\swarrow}$ *with* $T(\alpha.a.\beta.b.\gamma) = T(\alpha.\beta_1.b.a.\beta_2.\gamma)$ *and* β_1 *a subword of* β.

The induction step exploits Lemma 3. We illustrate the lemma on an example.

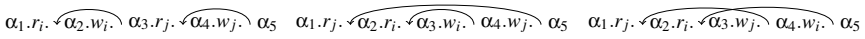
Example 5. For $\tau = r_y.\mathbf{w_z}.\mathbf{w_y}.\mathbf{f}.\mathbf{r_x}.w_x.w_z$, we have $r_y \rightarrow_{hb}^{+} w_x$ through $\mathbf{w_z}.\mathbf{w_y}.\mathbf{f}.\mathbf{r_x}$. As r_y and w_x both belong to program p_1, there is no alternative shuffle. But $T(\tau) = T(\tilde{\tau})$ with $\tilde{\tau} = \mathbf{w_z}.r_y.\mathbf{w_y}.\mathbf{f}.\mathbf{r_x}.w_x.w_z$.

3.2 From Many to Few: Locality of Reorderings

Reconsider the minimal violation $\alpha.r.\beta.w.\gamma$ where read r has overtaken write w. The following argumentation appears in variants several times. Lemma 5 gives $r \rightarrow_{hb}^{+} w$ through β. But as r has overtaken w, we also have the program order $w \rightarrow_{po}^{+} r$ and thus $w \rightarrow_{hb}^{+} r$. This means we found the happens before cycle $r \rightarrow_{hb}^{+} w \rightarrow_{hb}^{+} r$. The computation violates robustness by Lemma 1. A first consequence of this reasoning is Lemma 6. In a minimal violation, only a single program employs reorderings.

Lemma 6. *Consider the minimal violating computation* $\tau \in (\sqcup_{p\in\mathcal{P}}\tau_p)_{\checkmark}$ *with shortest derivations* $\sigma_p \curvearrowright^* \tau_p$. *There is precisely one component with* $\tau_p \neq \sigma_p$.

Proof. Assume there are at least two computations $\tau_p \neq \sigma_p$. By Lemma 4, all these computations contain a write w_p and a read r_p that overtook it, $w_p \leq_{\sigma_p} r_p$ and $r_p \leq_{\tau_p} w_p$. For the sake of readability, we use i, j as indices instead of programs p, p'. We single out the latest writes of distinct programs in τ that have been overtaken. Let them be $w_i \in \mathcal{A}(\tau_i)$ and $w_j \in \mathcal{A}(\tau_j)$ with $i \neq j$. For each of the writes, we find the latest read $r_i \in \mathcal{A}(\tau_i)$ and $r_j \in \mathcal{A}(\tau_j)$ that overtook it. There are three possible interactions:



Consider $\tau = \alpha_1.r_j.\alpha_2.r_i.\alpha_3.w_j.\alpha_4.w_i.\alpha_5$. The two cases left are simpler. By maximality of w_i, no write of α_5 has been overtaken. Similarly, by maximality of w_j no write of α_4, except for those of p_i, has been overtaken. We delete α_5 and the α_4 actions outside p_i:

$$\tau' = \alpha_1.r_j.\alpha_2.r_i.\alpha_3.w_j.(\alpha_4 \uparrow p_i).w_i.$$

The result is a TSO computation. It is feasible as r_i has overtaken $\alpha_4 \uparrow p_i$ and thus $\alpha_4 \uparrow p_i$ only contains write actions. To see that τ' is violating, note that Lemma 5 gives $r_j \rightarrow^+_{hb} w_j$ through $\alpha_2.r_i.\alpha_3$ of τ. Computation $\alpha_2.r_i.\alpha_3$ does not change in τ' and so $r_j \rightarrow^+_{hb} w_j$ carries over to τ'. With $w_j \rightarrow^+_{po} r_j$ and Lemma 1, the trace of τ' is not SC.

As the reorderings of τ and τ' coincide, also τ' is a minimal violating computation. By Lemma 5, $r_i \rightarrow^+_{hb} w_i$ through $\alpha_3.w_j.(\alpha_4 \uparrow p_i)$. By the overtake of r_j, $\alpha_3.w_j$ only contains writes of p_j. A program order $c \rightarrow^+_{po} d$ between such writes only reaches reads originally located between them. So r_j is not on the path through $\alpha_3.w_j.(\alpha_4 \uparrow p_i)$. By maximality, r_j is the last action in σ_j after removal of α_4 and α_5. We delete r_j and get

$$\tau'' = \alpha_1.\alpha_2.r_i.\alpha_3.w_j.(\alpha_4 \uparrow p_i).w_i.$$

This TSO computation is feasible. It is violating by $r_i \rightarrow^+_{hb} w_i \rightarrow^+_{hb} r_i$. As the reordering of r_j over w_j is missing, τ'' is smaller than τ. A contradiction to minimality. □

With Lemma 6, a minimal violation has the form $\tau \in (\tau_p \sqcup (\sqcup_{p'\in\mathcal{P}\setminus\{p\}}\sigma_{p'}))_{\checkmark}$ where $\sigma_p \curvearrowright^+ \tau_p$. We argue that reorderings are applied only locally in τ_p. There is an earliest write w_0 that is overtaken. This write determines the remaining reorderings as follows. In front of w_0, there is a block of reads ranging from the earliest read r_1 that overtook w_0 to the latest such read r_{n+1}. Action w_0 starts a block of writes $w_0 \ldots w_{n+1}$ so that r_i was originally located behind w_i. Minimality of τ forbids further reorderings.

Lemma 7. *Let* $\tau \in (\tau_p \sqcup (\sqcup_{p'\in\mathcal{P}\setminus\{p\}}\sigma_{p'}))_{\checkmark}$ *a minimal violation with shortest derivation* $\sigma_p \curvearrowright^+ \tau_p$. *Then* $\sigma_p = \sigma_1.\sigma_2.\sigma_3$ *and* $\tau_p = \sigma_1.\rho_p.\omega_p.\sigma_3$ *where* ρ_p *consists of reads and* ω_p *of writes. The last element* r *of* ρ_p *overtook the last element* w *of* ω_p, $w \leq_{\sigma_2} r$.

With $\rho_p = r_1 \ldots r_{n+1}$ and $\omega_p = w_0 \ldots w_{n+1}$, the minimal violation has the form

$$\tau = \alpha.\rho.\beta.\omega.\gamma \quad \text{where } \rho = r_1.\rho'.r_{n+1} \text{ and } \omega = w_0.\omega'.w_{n+1}. \tag{1}$$

The cycle $r_{n+1} \rightarrow^+_{hb} w_0 \rightarrow^+_{po} r_{n+1}$ through β, however, yields an even simpler violation.

Proposition 1. *If there is a violating computation, then there also is a minimal violation* $\alpha.\rho.\beta.\omega_p \in (\tau_p \sqcup (\sqcup\!\!\sqcup_{p' \in \mathcal{P}\setminus\{p\}} \sigma_{p'}))_\checkmark$ *with* ρ *and* ω_p *as defined in (1).*

Example 6. Computation $r_y.\mathbf{w_z}.\mathbf{w_y}.\mathbf{f}.\mathbf{r_x}.w_x.w_z$ is a minimal violation of the above shape: $\alpha = \varepsilon$, $\rho = \rho_p = r_y$, $\beta = \mathbf{w_z}.\mathbf{w_y}.\mathbf{f}.\mathbf{r_x}$, and $\omega_p = w_x.w_z$.

3.3 From Few to SC: Variable Accesses and Main Result

We focus on violations $\alpha.\rho.\beta.\omega_p$ of the form in Proposition 1. Our goal is to establish happens before relations through β and ρ. With these relations, we conclude disjointness of certain variable accesses in ω_p and ρ. As a consequence, we decompose the violating TSO computation into *two feasible SC computations* — our main result.

To begin with, note that $r_{n+1} \rightarrow_{hb}^{+} w_0$ through β. We manipulate the computation to let β start and end with actions on this path. For example, we change $r_y.\mathbf{w_z}.\mathbf{w_y}.\mathbf{f}.\mathbf{r_x}.w_x.w_z$ to $\mathbf{w_z}.r_y.\mathbf{w_y}.\mathbf{f}.\mathbf{r_x}.w_x.w_z$ where $\beta = \mathbf{w_y}.\mathbf{f}.\mathbf{r_x}$ directly connects r_y and w_x.

Lemma 8. *If there is a violation, there is a minimal violation* $\alpha.\rho.\beta.\omega_p$ *with* $\beta = a.\beta'.b$ *so that* $r_{n+1} \rightarrow_{hb} a \rightarrow_{hb}^{+} b \rightarrow_{hb} w_0$.

The idea is to shuffle all actions not in happens before with r_{n+1} to the front. Lemma 9 shows this does not change the trace. Set $HB(a,\beta) := \{b \in \mathcal{A}(\beta) \mid a \rightarrow_{hb}^{+} b \text{ through } \beta\}$.

Lemma 9. *Let* $\tau_1 = \alpha.a.\beta.\gamma$ *a minimal violation in* $(\sqcup\!\!\sqcup_{p\in\mathcal{P}} \tau_p)_\checkmark$. *There is an alternative shuffle* $\tau_2 = \alpha.\beta_1.a.\beta_2.\gamma$ *in* $(\sqcup\!\!\sqcup_{p\in\mathcal{P}} \tau_p)_\checkmark$ *with* $T(\tau_1) = T(\tau_2)$ *so that* $\mathcal{A}(\beta_2) = HB(a,\beta)$.

Consider a minimal violation $\alpha.\rho.\beta.\omega_p$ from Lemma 8 with

$$\rho = r_1.\rho_1 \ldots r_n.\rho_n.r_{n+1} \qquad \omega_p = w_0.\omega_1.w_1 \ldots \omega_{n+1}.w_{n+1}. \tag{2}$$

Here, r_i are the reordered reads of program p, and w_i is the first write that r_i overtook. The sequences ρ_i contain actions from components different from p. With Lemma 9, we can assume that for every $1 \le i \le n$ and every $c \in \mathcal{A}(\rho_i)$ we have $r_i \rightarrow_{hb}^{+} c$. These relations allow for precise statements about the variables accessed within the ρ_i.

Lemma 10. *Consider a minimal violating computation* $\alpha.\rho.\beta.\omega_p$ *of the form (2) above. For every* $c \in \mathcal{A}(\rho_i)$ *and* $w \in \mathcal{A}(\omega_j.w_j)$ *with* $j \le i$, *we have* $var(c) \neq var(w)$.

We now state our first main result. A violating TSO computation leads to a *pair of conflicting SC computations*

$$\sigma_1 = \alpha.\rho_1 \ldots \rho_n.a.\beta.b \qquad \sigma_2 = \alpha.w.\omega_1.\rho_1 \ldots \omega_n.\rho_n.\omega_{n+1}.r$$

that obey four constraints. (Prog) Computation $w.\omega_1 \ldots \omega_{n+1}.r$ belongs to a program $p \in \mathcal{P}$, the actions in $\rho_1 \ldots \rho_n.a.\beta.b$ belong to programs different from p. (Shuffle) The actions in ρ_i do not access variables written in ω_j or w with $j \le i$. (Hb) There is a path $a \rightarrow_{hb}^{+} b$ through β, $var(a) = var(r)$, $var(b) = var(w)$, $com(a) = w = com(w)$. (Reorder) There is a reordering of r over w.

Theorem 1. *There is a violating TSO computation if and only if there is a pair of conflicting SC computations.*

Example 7. The minimal violation $\mathbf{w_z}.r_y.\mathbf{w_y}.\mathbf{f}.\mathbf{r_x}.w_x.w_z$ computed above is of the form in Lemma 10. We set $\sigma_1 = \mathbf{w_z}.\mathbf{w_y}.\mathbf{f}.\mathbf{r_x}$ and $\sigma_2 = \mathbf{w_z}.w_x.w_z.r_z.r_y$.

It is known that data race freedom guarantees robustness. Theorem 1 strengthens this: a program is robust if and only if there are no SC computations with crossing data races.

4 Complexity of Robustness

To prove **Rob** in PSPACE, we find bounds on the length of the conflicting computations. The idea is to augment computations by information about the states of the components. This leads to a notion of configuration, and we prune a computation when it repeats configurations. The analysis reveals the data structures required in the decision procedure.

A *configuration* is a pair (s,m) consisting of a vector $s = (q_1,\ldots,q_n)$ of states of the components and a memory valuation m. An SC computation $\sigma \in (\sqcup\!\sqcup_{p\in\mathcal{P}}\sigma_p)_\checkmark$ with $\sigma = a_1.a_2\ldots$ defines transitions $(s_0,m_0).a_1.(s_1,m_1).a_2\ldots$ between configurations in the expected way. Let the union of the states of all components be $Q = Q_1 \cup \ldots \cup Q_n$. As the number of configurations is bounded by $2^{|Q|+|\mathcal{V}|}$, a computation α in σ_1 and σ_2 that is longer repeats configurations $(s_i,m_i) = (s_j,m_j)$. We cut out $a_{i+1}\ldots a_j$ and get $\alpha' = a_1\ldots a_i.a_{j+1}\ldots$ The computation is feasible as $(s_i,m_i) = (s_j,m_j)$ hold the states.

Lemma 11. *If there is a pair of conflicting SC computations, then there is one where the length* $|\alpha|$ *is bounded by* $2^{|Q|+|\mathcal{V}|}$.

For $\omega_1.\rho_1\ldots\omega_n.\rho_n.\omega_{n+1}$ in σ_2 pruning is more delicate. We have to ensure the result of pruning $\rho_1\ldots\rho_n$ still enables $a.\beta.b$ in σ_1. To this end, we guarantee that the configuration reached with the instructions in $\rho_1\ldots\rho_n$ remains unchanged. Moreover, we need to ensure that, after pruning, the reads in $\omega_1\ldots\omega_{n+1}$ can still perform prefetches. Otherwise (Reorder) may fail for the short computation.

To respect these two constraints, we define *extended configurations* $(s,m,m_{\neq p},m_p)$. Valuation $m_{\neq p}$ is updated only by $\rho_1\ldots\rho_n$ or, equivalently, by instructions in programs different from p. Valuation m_p gives the last write of p to each variable. It is therefore three valued. Besides 0 and 1, m_p assigns $\perp$ if the variable has not been written. The valuation ensures that pruning preserves the prefetching and reordering capabilities. In fact, read $r = (r,x,v,p)$ in $\sigma.r$ can prefetch a write in σ iff $m_p(x) = v$ with m_p obtained from executing σ. The read can overtake σ iff $m_p(x) = \perp$, no write in σ accessed x.

We start with configuration $(s,m,m,m_\perp)$ where (s,m) is reached by α and $m_\perp(x) = \perp$ for all $x \in \mathcal{V}$. To define transitions $(s,m,m_{\neq p},m_p).a.(s',m',m'_{\neq p},m'_p)$ between extended configurations, consider state $s(p') = q$ and transition $q.a.q'$. We update s and m like standard configurations. The valuations $m_{\neq p}$ and m_p only react to writes $a = (w,x,v,p')$. We set $m'_{\neq p} = m_{\neq p}$ if $p' = p$ and set $m'_{\neq p} = m_{\neq p}\{v/x\}$ otherwise. For m_p, we set $m'_p = m_p$ if $p' \neq p$. In case $p' = p$, we get $m'_p = m_p\{v/x\}$. Reads and fences do not change the additional valuations.

Example 8. Consider $\sigma_2 = \mathbf{w_z}.w_x.w_z.r_z.r_y$. After execution of $\alpha = \mathbf{w_z}$, we start with $((q_0,\tilde{q}_1),m,m,m_\perp)$ where $m(x) = m(y) = m(z) = 0$. Write w_x yields the transition $((q_0,\tilde{q}_1),m,m,m_\perp).w_x.((q_1,\tilde{q}_1),m',m,m'_p)$ where the resulting extended configuration

has memory valuations $m'(x) = 1, m'(y) = m'(z) = 0$ and $m'_p(x) = 1, m'_p(y) = m'_p(z) = \perp$.

If computation $\omega_1.\rho_1 \dots \omega_n.\rho_n.\omega_{n+1}$ repeats two configurations $(s_i, m_i, m_{\neq p,i}, m_{p,i}) = cf = (s_j, m_j, m_{\neq p,j}, m_{p,j})$ in

$$(s, m, m, m_\perp).w.\omega_1 \dots \omega_{i1}.cf.\omega_{i2} \dots \omega_{j1}.cf.\omega_{j2} \dots$$

we cut out the intermediary actions $\omega_{i2} \dots \omega_{j1}$. We argue that this results in a pair of conflicting computations. By $m_{\neq p,i} = m_{\neq p,j}$, also $\rho_1 \dots \rho_{i-1}.\rho_j \dots \rho_n$ enables β. For (Reorder), the critical point is that a read $\omega_{j2} \dots (r, x, v, p) \dots$ may prefetch a write $\omega_{i2} \dots (w, x, v, p) \dots \omega_{j1}$ that is cut out. By definition of TSO rewriting, $(w, x, v, p) \dots \omega_{j1}$ does not contain another write to x. Hence, we have $m_{p,j}(x) = v$. Since $m_{p,j} = m_{p,i}$, also $m_{p,i}(x) = v$. This means the last write to x in $\omega_1 \dots \omega_{i1}$ also gave v. Hence, (r, x, v, p) can still prefetch v in the shorter sequence. If the prefetches are preserved, a read that can overtake all writes in the original computation, can do so in the pruned one.

Lemma 12. *If there is a pair of conflicting computations, then there is one where the length* $|\omega_1.\rho_1 \dots \omega_n.\rho_n.\omega_{n+1}|$ *is bounded by* $2^{|Q|+4|\mathcal{V}|}$.

When we prune β in $a.\beta.b$, we have to preserve the happens before path $a \rightarrow_{hb}^{+} b$. Fix a path $a \rightarrow_{hb} a_1 \rightarrow_{hb} \dots \rightarrow_{hb} a_n \rightarrow_{hb} b$. If it is longer than $2^{|Q|+|\mathcal{V}|}|\mathcal{A}|$, two actions $a_i = a_j$ as well as their predecessor configurations $(s_i, m_i) = (s_j, m_j)$ coincide:

$$\dots (s_i, m_i).a_i.(s_{i+1}, m_{i+1}) \dots (s_j, m_j).a_j \dots$$

Cutting out $(s_{i+1}, m_{i+1}) \dots (s_j, m_j).a_j$ gives a new computation β'. It is feasible as the configurations coincide. A happens before path $a \rightarrow_{hb}^{+} b$ exists, because $a_j \rightarrow_{hb} a_{j+1}$ entails $a_i \rightarrow_{hb} a_{j+1}$.

Lemma 13. *If there is a pair of conflicting computations, then there is one where the length* $|\beta|$ *is bounded by* $2^{2(|Q|+|\mathcal{V}|)}|\mathcal{A}|$ *with* $|\mathcal{A}| \leq 6|Q||\mathcal{V}|$.

We roughly overapproximated the number of components by the number of states. The additional factor in the exponent stems from the fact that between $a_k \rightarrow_{hb} a_{k+1}$ there may be at most $2^{|Q|+|\mathcal{V}|}$ instructions. The reasoning is like in the case of α.

Theorem 2. Rob *is in* PSPACE.

The following non-deterministic algorithm checks for violations in space linear in $|Q|$ and $|\mathcal{V}|$. It first finds computation σ_2. When it guesses the occurrence of w, the extended configurations $(s, m, m_{\neq p}, m_p)$ above keep track of the effect of ρ_i actions. They further allow us to check the reordering capabilities (Reorder) and disjointnesss between ρ_i and ω_j as required by (Shuffle). Eventually, the algorithm ends σ_2. As α and $\rho_1 \dots \rho_n$ coincide for σ_1 and σ_2, the states of the component programs are correct for σ_1. The state of $prog(w)$ is unimportant as $a.\beta.b$ does not use this program. The valuation for σ_1 is stored in $m_{\neq p}$. To find $a.\beta.b$ with $a \rightarrow_{hb}^{+} b$, we safe the last action c on the happens before path. If we guess that a next action d should be on the path, we compare it with c. If the programs coincide or the condition in Lemma 2 is met, we conclude $c \rightarrow_{hb}^{+} d$. Binary counters stop the algorithm when it exceeds the necessary length for σ_1 and σ_2.

For PSPACE-hardness of robustness, we reduce the problem of whether a state $q \in Q$ is reachable in a program $p = (Q, q_0, \rightarrow)$ [SC85]. Let the variables x and y be fresh for p. The idea is to append Dekker's protocol in Example 1 to the state q of interest in a way that a violation happens only in this gadget. To this end, we add one component of the protocol to q, the second component yields a new program p_2:

add to p $\quad q \xrightarrow{w_x} q_1 \xrightarrow{ry} cs$ $\qquad$ add as new program $\quad p_2 : \quad \tilde{q}_0 \xrightarrow{\mathbf{w_y}} \tilde{q}_1 \xrightarrow{\mathbf{r_x}} \tilde{cs}.$

We then append a fence to every action in the modified program p. Let the result be p'. State q is reachable in p if and only if $\{p', p_2\}$ admits a violating computation.

Theorem 3. Rob *is* PSPACE*-complete.*

5 Conclusion and Future Work

This paper provides the theoretical foundations for checking robustness against TSO. We proved the problem PSPACE-complete, thereby discovering surprising facts about TSO. There are always minimal violating computations that use reorderings only in a single component. They can be reflected by pairs of conflicting SC computations.

We believe these findings have a high practical impact. We intend to reduce the search for conflicting computations to reachability in SC. Then the techniques and tools for SC verification can be reused for robustness analysis. A second application of our results is monitoring. Instead of simulating buffers with high memory requirements, these algorithms should look for conflicts already in SC. Our results may also be exploited for program repair. Minimal violations reveal the fences that ensure robustness.

Acknowledgements. We thank Mohamed Faouzi Atig for helpful discussions. This work was supported by ANR-06-SETI-001 AVERISS and ANR-09-SEGI-016 VERIDYC.

References

[ABBM10] Atig, M.F., Bouajjani, A., Burckhardt, S., Musuvathi, M.: On the verification problem for weak memory models. In: POPL, pp. 7–18. ACM, New York (2010)

[AG96] Adve, S., Gharachorloo, K.: Shared memory consistency models: a tutorial. Computer 29(12), 66–76 (1996)

[Alg10] Alglave, J.: A Shared Memory Poetics. PhD thesis, University Paris 7 (2010)

[AMP96] Alur, R., Mcmillan, K., Peled, D.: Model-checking of correctness conditions for concurrent objects. In: LICS, pp. 219–228. IEEE Computer Society Press, Los Alamitos (1996)

[AMSS10] Alglave, J., Maranget, L., Sarkar, S., Sewell, P.: Fences in weak memory models. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 258–272. Springer, Heidelberg (2010)

[BAM07] Burckhardt, S., Alur, R., Martin, M.: Checkfence: checking consistency of concurrent data types on relaxed memory models. In: PLDI, pp. 12–21. ACM, New York (2007)

[BM08] Burckhardt, S., Musuvathi, M.: Effective program verification for relaxed memory models. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 107–120. Springer, Heidelberg (2008)

[BSS11] Burnim, J., Stergiou, C., Sen, K.: Sound and complete monitoring of sequential consistency for relaxed memory models. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 11–25. Springer, Heidelberg (2011)

[GK97] Gibbons, P.B., Korach, E.: Testing shared memories. SIAM J. Comp. 26(4), 1208–1244 (1997)

[KVY10] Kuperstein, M., Vechev, M., Yahav, E.: Automatic inference of memory fences. In: FMCAD, pp. 111–119. IEEE Computer Society Press, Los Alamitos (2010)

[Lam79] Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Trans. Comp. 28(9), 690–691 (1979)

[OSS09] Owens, S., Sarkar, S., Sewell, P.: A better x86 memory model: x86-TSO (extended version). Technical Report UCAM-CL-TR-745, Univ. of Cambridge (2009)

[Owe10] Owens, S.: Reasoning about the implementation of concurrency abstractions on x86-TSO. In: D'Hondt, T. (ed.) ECOOP 2010. LNCS, vol. 6183, pp. 478–503. Springer, Heidelberg (2010)

[SC85] Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. JACM 32(3), 733–749 (1985)

[SS88] Shasha, D., Snir, M.: Efficient and correct execution of parallel programs that share memory. ACM TOPLAS 10(2), 282–312 (1988)

[WG94] Weaver, D., Germond, T. (eds.): The SPARC Architecture Manual Version 9. PTR Prentice Hall, Upper Saddle River (1994)

Multiply-Recursive Upper Bounds with Higman's Lemma

S. Schmitz and Ph. Schnoebelen

LSV, ENS Cachan & CNRS, Cachan, France
{schmitz,phs}@lsv.ens-cachan.fr

Abstract. We develop a new analysis for the length of controlled bad sequences in well-quasi-orderings based on Higman's Lemma. This leads to tight multiply-recursive upper bounds that readily apply to several verification algorithms for well-structured systems.

1 Introduction

Well-quasi-orderings (wqo's) are an important tool in logic and computer science [13]. They are the key ingredient to a large number of decidability (or finiteness, regularity, ...) results. In constraint solving, automated deduction, program analysis, and many more fields, wqo's usually appear under the guise of specific tools, like Dickson's Lemma (for tuples of integers), Higman's Lemma (for words and their subwords), Kruskal's Tree Theorem and its variants (for finite trees with embeddings), and recently the Robertson-Seymour Theorem (for graphs and their minors). In program verification, wqo's are the basis for *well-structured systems* [1, 10, 11], a generic framework for infinite-state systems.

Complexity. Wqo's are seldom used in complexity analysis. In order to extract complexity upper bounds for an algorithm whose termination proof rests on Dickson's or Higman's Lemma, one must be able to bound the length of so-called "controlled bad sequences" (see Def. 2.4). Here the available results are not very well known in computer science, and their current packaging does not make them easy to read and apply. For applications like the complexity of lossy channel systems [4] that rely on Higman's Lemma over Γ_p^* (the words over a p-letter alphabet), what we really need is something like:

Length Function Theorem. *Let $L_{\Gamma_p^*}(n)$ be the maximal length of bad sequences $w_0, w_1, w_2, \ldots$ over Γ_p^* with $p \geq 2$ s.t. $|w_i| < g^i(n)$ for $i = 0, 1, 2, \ldots$ If the* control function *g is primitive-recursive, then the* length function *$L_{\Gamma_p^*}$ is bounded by a function in $\mathscr{F}_{\omega^{p-1}}$.*[1]

Unfortunately, the literature contains no such clear statement (see the comparison with existing work below).

[1] Here the functions F_α are the ordinal-indexed levels of the Fast-Growing Hierarchy [14], with multiply-recursive complexity starting at level $\alpha = \omega$, i.e., Ackermannian complexity, and stopping just before level $\alpha = \omega^\omega$, i.e., hyper-Ackermannian complexity. The function classes $\mathscr{F}_\alpha$ denote their elementary-recursive closure.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 441–452, 2011.

Our Contribution. We provide a new and *self-contained* proof of the Length Function Theorem, a fundamental result that (we think) deserves a wide audience. The exact statement we prove, Thm. 5.3 below, is rather general: it is parameterized by the control function g and accomodates various combinations of Γ_p^* sets without losing precision. For this we significantly extend the setting we developed for Dickson's Lemma [9]: We rely on iterated residuations with a simple but explicit algebraic framework for handling wqo's and their residuals in a compositional way. Our computations can be kept relatively simple by means of a fully explicit notion of "normed reflection" that captures the over-approximations we use, all the while enjoying good algebraic properties. We also show *how to apply* the Length Function Theorem by deriving precise multiply-recursive upper bounds, parameterized by the alphabet size, for the complexity of lossy channel systems and the Regular Post Embedding Problem (see Sec. 6).

Comparison with Existing Work. (Here, and for easier comparison, we assume that the control function g is the successor function.)

For $\mathbb{N}^k$ (i.e., Dickson's Lemma), Clote gives an explicit upper bound at level $\mathscr{F}_{k+6}$ extracted from complex Ramsey-theoretical results, hence hardly self-contained [8]. This is a simplification over an earlier analysis by McAloon, which leads to a uniform upper bound at level $\mathscr{F}_{k+1}$, but gives no explicit statement nor asymptotic analysis [15]. Both analyses are based on large intervals and extractions, and McAloon's is technically quite involved. With D. and S. Figueira, we improved this to an explicit and tight $\mathscr{F}_k$ [9].

For Γ_p^* (Higman's Lemma), Cichoń and Tahhan Bittar exhibit a reduction method, deducing bounds (for tuples of) words on Γ_p from bounds on the Γ_{p-1} case [7]. Their decomposition is clear and self-contained, with the control function made explicit. It ends up with some inequalities, collected in [7, Sec. 8], from which it is not clear what precisely are the upper bounds one can extract. Following this, Touzet claims a bound of F_{ω^p} [19, Thm. 1.2] with an analysis based on iterated residuations but the proof (given in [18]) is incomplete.

Finally, Weiermann gives an $\mathscr{F}_{\omega^{p-1}}$-like bound for Γ_p^* [20, Coro. 6.3] for sequences produced by term rewriting systems, but his analysis is considerably more involved (as can be expected since it applies to the more general Kruskal Theorem) and one cannot easily extract an explicit proof for his Coro. 6.3.

Regarding lower bounds, it is known that $F_{\omega^{p-1}}$ is essentially tight [6].

Outline of the Paper. All basic notions are recalled in Sec. 2, leading to the Descent Equation (3). Reflections in an algebraic setting are defined in Sec. 3, then transfered in an ordinal-arithmetic setting in Sec. 4. We prove the Main Theorem in Sec. 5, before illustrating its uses in Sec. 6. All the proofs missing from this extended abstract can be found at `http://arxiv.org/abs/1103.4399`.

2 Normed Wqo's and Controlled Bad Sequences

We recall some basic notions of wqo-theory [see e.g. 13]. A *quasi-ordering* (a "qo") is a relation $(A; \leq)$ that is reflexive and transitive. As usual, we write

$x < y$ when $x \leq y$ and $y \not\leq x$, and we denote structures $(A; P_1, \ldots, P_m)$ with just the support set A when this does not lead to ambiguities. Classically, the substructure *induced* by a subset $X \subseteq A$ is $(X; P_{1|X}, \ldots, P_{m|X})$ where, for a predicate P over A, $P_{|X}$ is its trace over X.

A qo A is a *well-quasi-ordering* (a "wqo") if every infinite sequence $x_0, x_1, x_2, \ldots$ contains an infinite increasing subsequence $x_{i_0} \leq x_{i_1} \leq x_{i_2} \cdots$ Equivalently, a qo is a wqo if it is well-founded (has no infinite strictly decreasing sequences) and contains no infinite antichains (i.e., set of pairwise incomparable elements). Every induced substructure of a wqo is a wqo.

Wqo's With Norms. A *norm function* over a set A is a mapping $|.|_A : A \to \mathbb{N}$ that provides every element of A with a positive integer, its *norm*, capturing a notion of size. For $n \in \mathbb{N}$, we let $A_{<n} \stackrel{\text{def}}{=} \{x \in A \mid |x|_A < n\}$ denote the subset of elements with norm below n. The norm function is said to be *proper* if $A_{<n}$ is finite for every n.

Definition 2.1. *A* normed wqo *(a "nwqo") is a wqo* $(A; \leq_A, |.|_A)$ *equipped with a proper norm function.*

There are no special conditions on norms, except being proper. In particular no connection is required between the ordering of elements and their norms. In applications, norms are related to natural complexity measures.

Example 2.2 (Some Basic Wqo's). The set of natural numbers $\mathbb{N}$ with the usual ordering is the smallest infinite wqo. For every $p \in \mathbb{N}$, we single out two p-element wqo's: $\searrow_p$ is the p-element initial segment of $\mathbb{N}$, i.e., the set $\{0, 1, 2, \ldots, p-1\}$ ordered linearly, while Γ_p is the p-letter alphabet $\{a_1, \ldots, a_p\}$ where distinct letters are unordered. We turn them into nwqo's by fixing the following:

$$|k|_{\mathbb{N}} = |k|_{\searrow_p} \stackrel{\text{def}}{=} k\,, \qquad\qquad |a_i|_{\Gamma_p} \stackrel{\text{def}}{=} 0\,. \tag{1}$$

We write $A \equiv B$ when the two nwqo's A and B are *isomorphic* structures. For all practical purposes, isomorphic nwqo's can be identified, following a standard practice that significantly simplifies the notational apparatus we develop in Sec. 3. For the moment, we only want to stress that, in particular, *norm functions must be preserved* by nwqo isomorphisms.

Example 2.3 (Isomorphism Between Basic Nwqo's). On the positive side, $\searrow_0 \equiv \Gamma_0$ and also $\searrow_1 \equiv \Gamma_1$ since $|a_1|_{\Gamma_1} = 0 = |0|_{\searrow_1}$. By contrast $\searrow_2 \not\equiv \Gamma_2$: not only these two have non-isomorphic order relations, they also have different norm functions.

Good, Bad, and Controlled Sequences. A sequence $\mathbf{x} = x_0, x_1, x_2, \ldots$ over a qo is *good* if $x_i \leq x_j$ for some positions $i < j$. It is *bad* otherwise. *Over a wqo*, all infinite sequences are good (equivalently, all bad sequences are finite).

We are interested in the maximal length of bad sequences for a given wqo. Here, a difficulty is that, in general, bad sequences can be arbitrarily long and there is no finite maximal length. However, in our applications we are only interested in bad sequences generated by some algorithmic method, i.e., bad sequences whose complexity is controlled in some way.

Definition 2.4 (Control Functions and Controlled Sequences) *A* control function *is a mapping* $g : \mathbb{N} \to \mathbb{N}$. *For a* size $n \in \mathbb{N}$, *a sequence* $\mathbf{x} = x_0, x_1, x_2, \ldots$ *over a nwqo* A *is* (g, n)-controlled $\overset{\text{def}}{\Leftrightarrow}$

$$\forall i = 0, 1, 2, \ldots : \; |x_i|_A < g^i(n) \;=\; \overbrace{g(g(\ldots g}^{i \text{ times}}(n))) \,.$$

Why n is called a "size" appears with Prop. 2.8 and its proof. A pair (g, n) is just called a *control* for short. We say that a sequence $\mathbf{x}$ is n-*controlled* (or just *controlled*), when g (resp. g and n) is clear from the context. Observe that the empty sequence is always a controlled sequence.

Proposition 2.5. *Let* A *be a nwqo and* (g, n) *a control. There exists a finite maximal length* $L \in \mathbb{N}$ *for* (g, n)*-controlled bad sequences over* A.

We write $L_{A,g}(n)$ for this maximal length, a number that depends on all three parameters: A, g and n. However, for complexity analysis, the relevant information is how, for given A and g, the *length function* $L_{A,g} : \mathbb{N} \to \mathbb{N}$ behaves asymptotically, hence our choice of notation. Furthermore, g is a parameter that remains fixed in our analysis and applications, hence it is usually left implicit. **From now on we assume a fixed control function** g and just write $L_A(n)$ for $L_{A,g}(n)$. We further assume that g is *smooth* ($\overset{\text{def}}{\Leftrightarrow}$ $g(x+1) \geq g(x)+1 \geq x+2$ for all x), which is harmless for applications but simplifies computations like (4).

Residuals Wqo's and a Descent Equation. Via residuals one expresses the length function by induction over nwqo's.

Definition 2.6 (Residuals). *For a nwqo* A *and an element* $x \in A$, *the* residual A/x *is the substructure (a nwqo) induced by the subset* $A/x \overset{\text{def}}{=} \{y \in A \mid x \not\leq y\}$ *of elements that are not above* x.

Example 2.7 (Residuals of Basic Nwqo's). For all $k < p$ and $i = 1, \ldots, p$:

$$\mathbb{N}/k = \searrow_p/k = \searrow_k \,, \qquad \Gamma_p/a_i \equiv \Gamma_{p-1} \,. \tag{2}$$

Proposition 2.8 (Descent Equation)

$$L_A(n) = \max_{x \in A_{<n}} \left\{1 + L_{A/x}(g(n))\right\} . \tag{3}$$

This reduces the L_A function to a finite combination of L_{A_i}'s where the A_i's are residuals of A, hence "smaller" sets. Residuation is well-founded for wqo's: a sequence of successive residuals $A \supsetneq A/x_0 \supsetneq A/x_0/x_1 \supsetneq A/x_0/x_1/x_2 \supsetneq \cdots$ is necessarily finite since $x_0, x_1, x_2, \ldots$ must be a bad sequence. Hence the recursion in the Descent Equation is well-founded and can be used to evaluate $L_A(n)$. This is our starting point for analyzing the behaviour of length functions.

For example, using induction and Eq. (2), the Descent Equation leads to:

$$L_{\Gamma_p}(n) = p \,, \qquad L_{\mathbb{N}}(n) = n \,, \qquad L_{\searrow_p}(n) = \min(n, p) \,. \tag{4}$$

3 An Algebra of Normed Wqo's

The algebraic framework we now develop has two main goals. First it provides a *notation* for denoting the wqo's encountered in algorithmic applications. These wqo's and their norm functions abstract data structures that are built inductively by combining some basic wqo's. Second, it supports a *calculus* for the kind of compositional computations, based on the Descent Equation, we develop next.

The constructions we use in this paper are disjoint sums, cartesian products, and Kleene stars (with Higman's order). These constructions are classic. Here we also have to define how they combine the norm functions:

Definition 3.1 (Sums, Products, Stars Nwqo's) *The* disjoint sum $A_1 + A_2$ *of two nwqos* A_1 *and* A_2 *is the nwqo given by*

$$A_1 + A_2 = \{\langle i, x\rangle \mid i \in \{1,2\} \text{ and } x \in A_i\}\,, \tag{5}$$

$$\langle i, x\rangle \leq_{A_1+A_2} \langle j, y\rangle \stackrel{\text{def}}{\Leftrightarrow} i = j \text{ and } x \leq_{A_i} y\,, \tag{6}$$

$$|\langle i, x\rangle|_{A_1+A_2} \stackrel{\text{def}}{=} |x|_{A_i}\,. \tag{7}$$

The cartesian product $A_1 \times A_2$ *of two nwqos* A_1 *and* A_2 *is the nwqo given by*

$$A_1 \times A_2 = \{\langle x_1, x_2\rangle \mid x_1 \in A_1, x_2 \in A_2\}\,, \tag{8}$$

$$\langle x_1, x_2\rangle \leq_{A_1\times A_2} \langle y_1, y_2\rangle \stackrel{\text{def}}{\Leftrightarrow} x_1 \leq_{A_1} y_1 \text{ and } x_2 \leq_{A_2} y_2\,, \tag{9}$$

$$|\langle x_1, x_2\rangle|_{A_1\times A_2} \stackrel{\text{def}}{=} \max(|x_1|_{A_1}, |x_2|_{A_2})\,. \tag{10}$$

The Kleene star A^* *of a nwqo* A *is the nwqo given by*

$$A^* \stackrel{\text{def}}{=} \text{all finite lists } (x_1 \dots x_n) \text{ of elements of } A\,, \tag{11}$$

$$(x_1 \dots x_n) \leq_{A^*} (y_1 \dots y_m) \stackrel{\text{def}}{\Leftrightarrow} \begin{cases} x_1 \leq_A y_{i_1} \wedge \dots \wedge x_n \leq_A y_{i_n} \\ \text{for some } 1 \leq i_1 < i_2 < \dots < i_n \leq m \end{cases}\,, \tag{12}$$

$$|(x_1 \dots x_n)|_{A^*} \stackrel{\text{def}}{=} \max(n, |x_1|_A, \dots, |x_n|_A)\,. \tag{13}$$

It is well-known (and plain) that $A_1 + A_2$ and $A_1 \times A_2$ are indeed wqo's when A_1 and A_2 are. The fact that A^* is a wqo when A is, is a classical result called Higman's Lemma. We let the reader check that the norm functions defined in Equations (7), (10), and (13), are proper and turn $A_1 + A_2$, $A_1 \times A_2$ and A^* into nwqo's. Finally, we note that nwqo isomorphism is a congruence for sum, product and Kleene star.

Notation (0 and 1) *We let* $\mathbf{0}$ *and* $\mathbf{1}$ *be short-hand notations for, respectively,* Γ_0 *(the empty nwqo) and* Γ_1 *(the singleton nwqo with the* 0 *norm).*

This is convenient for writing down the following algebraic properties:

Proposition 3.2 *The following isomorphisms hold:*

$$A + B \equiv B + A\,, \qquad A + (B + C) \equiv (A + B) + C\,,$$
$$A \times B \equiv B \times A\,, \qquad A \times (B \times C) \equiv (A \times B) \times C\,,$$
$$\mathbf{0} + A \equiv A\,, \qquad \mathbf{1} \times A \equiv A\,,$$
$$\mathbf{0} \times A \equiv \mathbf{0}\,, \qquad (A + A') \times B \equiv (A \times B) + (A' \times B)\,,$$
$$\mathbf{0}^* \equiv \mathbf{1}\,, \qquad \mathbf{1}^* \equiv \mathbb{N}\,.$$

In view of these properties, we freely write $A \cdot k$ and A^k for the k-fold sums and products $A + \cdots + A$ and $A \times \cdots \times A$. Observe that $A \cdot k \equiv A \times \Gamma_k$.

Reflecting Normed Wqo's. Reflections are the main comparison/abstraction tool we shall use. They let us simplify instances of the Descent Equation by replacing all A/x for $x \in A_{<n}$ by a single (or a few) A' that is smaller than A but large enough to reflect all considered A/x's.

Definition 3.3 *A* nwqo reflection *is a mapping $h : A \to B$ between two nwqo's that satisfies the two following properties:*

$$\forall x, y \in A : h(x) \leq_B h(y) \text{ implies } x \leq_A y \,,$$
$$\forall x \in A : |h(x)|_B \leq |x|_A \,.$$

In other words, a nwqo reflection is an order reflection that is also norm-decreasing (not necessarily strictly).

We write $h : A \hookrightarrow B$ when h is a nwqo reflection and say that B *reflects* A. This induces a relation between nwqos, written $A \hookrightarrow B$.

Reflection is transitive since $h : A \hookrightarrow B$ and $h' : B \hookrightarrow C$ entails $h' \circ h : A \hookrightarrow C$. It is also reflexive, hence reflection is a quasi-ordering. Any nwqo reflects its substructures since $Id : X \hookrightarrow A$ when X is a substructure of A. Thus $\mathbf{0} \hookrightarrow A$ for any A, and $\mathbf{1} \hookrightarrow A$ for any non-empty A.

Example 3.4 Among the basic nwqos from Example 2.2, we note the following relations (or absences thereof). For any $p \in \mathbb{N}$, $\searrow_p \hookrightarrow \Gamma_p$, while $\Gamma_p \not\hookrightarrow \searrow_p$ when $p \geq 2$. The reflection of substructures yields $\searrow_p \hookrightarrow \mathbb{N}$ and $\Gamma_p \hookrightarrow \Gamma_{p+1}$. Obviously, $\mathbb{N} \not\hookrightarrow \searrow_p$ and $\Gamma_{p+1} \not\hookrightarrow \Gamma_p$.

Reflections preserve controlled bad sequences. Let $h : A \hookrightarrow B$, consider a sequence $\mathbf{x} = x_0, x_1, \ldots, x_l$ over A, and write $h(\mathbf{x})$ for $h(x_0), h(x_1), \ldots, h(x_l)$, a sequence over B. Then $h(\mathbf{x})$ is bad when $\mathbf{x}$ is, and n-controlled when $\mathbf{x}$ is. Hence:

$$A \hookrightarrow B \text{ implies } L_A(n) \leq L_B(n) \text{ for all } n \,. \tag{14}$$

Reflections are compatible with product, sum, and Kleene star.

Proposition 3.5 (Reflection is a Preconguence)

$$A \hookrightarrow A' \text{ and } B \hookrightarrow B' \text{ imply } A + B \hookrightarrow A' + B' \text{ and } A \times B \hookrightarrow A' \times B' \,, \tag{15}$$
$$A \hookrightarrow A' \text{ implies } A^* \hookrightarrow A'^* \,. \tag{16}$$

Computing and Reflecting Residuals. We may now tackle our first main problem: computing residuals A/x. This is done by induction over the structure of A.

Proposition 3.6 (Inductive Rules For Residuals)

$$(A + B)/\langle 1, x\rangle = (A/x) + B \,, \quad (A + B)/\langle 2, x\rangle = A + (B/x) \,, \tag{17}$$
$$(A \times B)/\langle x, y\rangle \hookrightarrow \big[(A/x) \times B\big] + \big[A \times (B/y)\big] \,, \tag{18}$$
$$A^*/(x_1 \ldots x_n) \hookrightarrow \Gamma_n \times A^n \times (A/x_1)^* \times \cdots \times (A/x_n)^* \,, \tag{19}$$
$$\Gamma^*_{p+1}/(x_1 \ldots x_n) \hookrightarrow \Gamma_n \times (\Gamma^*_p)^n \,. \tag{20}$$

Equation (20) is a refinement of (19) in the case of finite alphabets.

Since it provides reflections instead of isomorphisms, Prop. 3.6 is not meant to support exact computations of A/x by induction over the structure of A. More to the point, it yields over-approximations that are sufficiently precise for our purposes while bringing important simplifications when we have to reflect (the max of) all A/x for all $x \in A_{<n}$.

4 Reflecting Residuals in Ordinal Arithmetic

We now introduce an *ordinal* notation for nwqo's. The purpose is twofold. Firstly, the ad-hoc techniques we use for evaluating, reflecting, and comparing residual nwqo's are more naturally stated within the language of ordinal arithmetic. Secondly, these ordinals will be essential for bounding L_A using functions in subrecursive hierarchies. For these developments, we restrict ourselves to *exponential* nwqo's, i.e., nwqo's obtained from finite Γ_p's with sums, products, and *Kleene star restricted to the* Γ_p*'s*. Modulo isomorphism, $\mathbb{N}^k \equiv \prod_{i=1}^{k} \Gamma_1^*$ is exponential.

Ordinal Terms. We use Greek letters like $\alpha, \beta, \ldots$ to denote ordinal terms in Cantor Normal Form (CNF) built using 0, addition, and ω-exponentiation (we restrict ourselves to ordinals $< \varepsilon_0$). A term α has the general form $\alpha = \omega^{\beta_1} + \omega^{\beta_2} + \cdots + \omega^{\beta_m}$ with $\beta_1 \geq \beta_2 \geq \cdots \geq \beta_m$ (ordering defined below) and where we distinguish between three cases: α is 0 if $m = 0$, α is a *successor* if ($m > 0$ and) $\beta_m = 0$, α is a *limit* if $\beta_m \neq 0$ (in the following, λ will always denote a limit, and we write $\alpha + 1$ rather than $\alpha + \omega^0$ for a successor). We say that α is principal (additive) if $m = 1$.

Ordering among our ordinals is defined inductively by

$$\alpha < \alpha' \overset{\text{def}}{\Leftrightarrow} \begin{cases} \alpha = 0 \text{ and } \alpha' \neq 0, \text{ or} \\ \alpha = \omega^\beta + \gamma,\ \alpha' = \omega^{\beta'} + \gamma' \text{ and } \begin{cases} \beta < \beta', \text{ or} \\ \beta = \beta' \text{ and } \gamma < \gamma'. \end{cases} \end{cases} \tag{21}$$

We let $\mathrm{CNF}(\alpha)$ denote the set of ordinal terms $<\alpha$.

For $c \in \mathbb{N}$, $\omega^\beta \cdot c$ denotes the c-fold addition $\omega^\beta + \cdots + \omega^\beta$. We sometimes write terms under a "strict" form $\alpha = \omega^{\beta_1} \cdot c_1 + \omega^{\beta_2} \cdot c_2 + \cdots + \omega^{\beta_m} \cdot c_m$ with $\beta_1 > \beta_2 > \cdots > \beta_m$, where the c_i's, called *coefficients*, must be > 0.

Recall the definitions of the *natural sum* $\alpha \oplus \alpha'$ and *natural product* $\alpha \otimes \alpha'$ of two terms in $\mathrm{CNF}(\varepsilon_0)$:

$$\sum_{i=1}^{m} \omega^{\beta_i} \oplus \sum_{j=1}^{n} \omega^{\beta'_j} \overset{\text{def}}{=} \sum_{k=1}^{m+n} \omega^{\gamma_k}, \qquad \sum_{i=1}^{m} \omega^{\beta_i} \otimes \sum_{j=1}^{n} \omega^{\beta'_j} \overset{\text{def}}{=} \bigoplus_{i=1}^{m} \bigoplus_{j=1}^{n} \omega^{\beta_i \oplus \beta'_j},$$

where $\gamma_1 \geq \cdots \geq \gamma_{m+n}$ is a rearrangement of $\beta_1, \ldots, \beta_m, \beta'_1, \ldots, \beta'_n$. For $\alpha \in \mathrm{CNF}(\omega^{\omega^\omega})$, the decomposition $\alpha = \sum_{i=1}^{m} \omega^{\beta_i}$ uses β_i's that are in $\mathrm{CNF}(\omega^\omega)$, i.e., of the form $\beta_i = \sum_{j=1}^{k_i} \omega^{p_{i,j}}$ (with each $p_{i,j} < \omega$) so that ω^{β_i} is $\bigotimes_{j=1}^{k_i} \omega^{\omega^{p_{i,j}}}$. A term ω^{ω^p} is called a *principal multiplicative*.

We map exponential nwqo's to ordinals in $\mathrm{CNF}(\omega^{\omega^\omega})$ using their *maximal order type* [12]. Formally $o(A)$ is defined by

$$o(\Gamma_p) \stackrel{\text{def}}{=} p\,, \qquad o(\Gamma_0^*) \stackrel{\text{def}}{=} \omega^0\,, \qquad o(\Gamma_{p+1}^*) \stackrel{\text{def}}{=} \omega^{\omega^p}\,, \tag{22}$$

$$o(A+B) \stackrel{\text{def}}{=} o(A) \oplus o(B)\,, \qquad o(A \times B) \stackrel{\text{def}}{=} o(A) \otimes o(B)\,. \tag{23}$$

Conversely, there is a *canonical exponential nwqo* $C(\alpha)$ for each α in $\mathrm{CNF}(\omega^{\omega^\omega})$:

$$C\Big(\omega^{\beta_1} + \cdots + \omega^{\beta_m}\Big) = C\Big(\bigoplus_{i=1}^{m} \bigotimes_{j=1}^{k_i} \omega^{\omega^{p_{i,j}}}\Big) \stackrel{\text{def}}{=} \sum_{i=1}^{m} \prod_{j=1}^{k_i} \Gamma^*_{(p_{i,j}+1)}\,. \tag{24}$$

Then, o and C are bijective inverses (modulo isomorphism of nwqo's), compatible with sums and products. This correspondence equates between terms that, on one side, denote partial orderings with norms, and on the other side, ordinals in $\mathrm{CNF}(\omega^{\omega^\omega})$.

Derivatives. We aim to replace the "all A/x for $x \in A_{<n}$" by a computation of "some derived $\alpha' \in \partial_n\alpha$" where $\alpha = o(A)$, see Thm. 4.1 below. For this purpose, the definition of derivatives is based on the inductive rules in Prop. 3.6.

Let $n > 0$ be some norm. We start with principal ordinals and define

$$D_n\Big(\omega^{\omega^p}\Big) \stackrel{\text{def}}{=} \begin{cases} n-1 & \text{if } p = 0, \\ \omega^{(\omega^{p-1}\cdot(n-1))} \cdot (n-1) & \text{otherwise.} \end{cases} \tag{25}$$

$$D_n\left(\omega^{\omega^{p_1}+\cdots+\omega^{p_k}}\right) \stackrel{\text{def}}{=} \bigoplus_{j=1}^{k} \left(D_n\Big(\omega^{\omega^{p_j}}\Big) \otimes \bigotimes_{\ell \neq j} \omega^{\omega^{p_\ell}} \right)\,. \tag{26}$$

Now, with any $\alpha \in \mathrm{CNF}(\omega^{\omega^\omega})$, we associate the set of its *derivatives* $\partial_n\alpha$ with

$$\partial_n\Big(\sum_{i=1}^{m} \omega^{\beta_i}\Big) \stackrel{\text{def}}{=} \Big\{ D_n\big(\omega^{\beta_i}\big) \oplus \sum_{\ell \neq i} \omega^{\beta_\ell} \;\Big|\; i = 1, \ldots, m \Big\}\,. \tag{27}$$

This yields, for example, and assuming $p, k > 0$:

$$D_n(1) = 0,\ D_n(\omega) = n-1,\ D_n\big(\omega^{\omega^p \cdot k}\big) = \omega^{[\omega^p\cdot(k-1)+\omega^{p-1}\cdot(n-1)]} \cdot k(n-1)\,, \tag{28}$$

$$\partial_n 0 = \emptyset,\ \partial_n 1 = \{0\},\ \partial_n\omega = \{n-1\},\ \partial_n(\omega^\beta \cdot (k+1)) = \{\omega^\beta \cdot k \oplus D_n(\omega^\beta)\}. \tag{29}$$

Thus $\partial_n\alpha$ can be a singleton even when α is not principal, e.g., $\partial_n(p+1) = \{p\}$. We sometimes write $\alpha\ \partial_n\ \alpha'$ instead of $\alpha' \in \partial_n\alpha$, seeing ∂_n as a relation. Note that $\partial_n\alpha \subseteq \mathrm{CNF}(\alpha)$, hence $\partial \stackrel{\text{def}}{=} \bigcup_{n<\omega} \partial_n$ is well-founded.

Theorem 4.1 (Reflection by Derivatives). *Let $x \in A_{<n}$ for some exponential A. Then there exists $\alpha' \in \partial_n o(A)$ s.t. $A/x \hookrightarrow C(\alpha')$.*

Combining with equations (3) and (14), we obtain:

$$L_{C(\alpha)}(n) \leq \max_{\alpha' \in \partial_n\alpha} \big\{1 + L_{C(\alpha')}(g(n))\big\}\,. \tag{30}$$

5 Classifying L Using Subrecursive Hierarchies

For α in $\mathrm{CNF}(\omega^{\omega^\omega})$, define

$$M_\alpha(n) \stackrel{\text{def}}{=} \max_{\alpha' \in \partial_n \alpha} \{1 + M_{\alpha'}(g(n))\} \ . \tag{31}$$

(Recall that ∂ is well-founded, thus (31) is well-defined). Comparing with (30), we see that M_α bounds the length function: $M_\alpha(n) \geq L_{C(\alpha)}(n)$.

This defines an ordinal-indexed family of functions $(M_\alpha)_{\alpha \in \mathrm{CNF}(\omega^{\omega^\omega})}$ similar to some classical subrecursive hierarchies, with the added twist of the max operation—see [2, 16] for somewhat similar hierarchies. This is a real issue and one cannot replace a "$\max_{\alpha \in \ldots}\{M_\alpha(x)\}$" with "$M_{\sup\{\alpha \in \ldots\}}(x)$" since M_α is not always bounded by $M_{\alpha'}$ when $\alpha < \alpha'$. E.g., $M_{n+2}(n) = n+2 > M_\omega(n) = n+1$.

Subrecursive Hierarchies have been introduced as generators of classes of functions. For instance, writing $\mathscr{F}_\alpha$ for the class of functions elementary-recursive in the function F_α of the *fast growing hierarchy*, we can characterize the set of primitive-recursive functions as $\bigcup_{k<\omega} \mathscr{F}_k$, or that of multiply-recursive functions as $\bigcup_{\beta<\omega^\omega} \mathscr{F}_\beta$ [14].

Let us introduce (slight generalizations of) several classical hierarchies from [14, 7]. Those hierarchies are defined through assignments of *fundamental sequences* $(\lambda_x)_{x<\omega}$ for limit ordinals $\lambda < \varepsilon_0$, verifying $\lambda_x < \lambda$ for all x and $\lambda = \sup_x \lambda_x$. A standard assignment is defined by:

$$(\gamma + \omega^{\beta+1})_x \stackrel{\text{def}}{=} \gamma + \omega^\beta \cdot (x+1)\,, \qquad (\gamma + \omega^\lambda)_x \stackrel{\text{def}}{=} \gamma + \omega^{\lambda_x}\,, \tag{32}$$

where γ can be 0. Note that, in particular, $\omega_x = x+1$. Given an assignment of fundamental sequences, one can define the (x-indexed) *predecessor* $P_x(\alpha) < \alpha$ of an ordinal $\alpha \neq 0$ as

$$P_x(\alpha+1) \stackrel{\text{def}}{=} \alpha\,, \qquad P_x(\lambda) \stackrel{\text{def}}{=} P_x(\lambda_x)\,. \tag{33}$$

Given a fixed smooth control function h, the *Hardy hierarchy* $(h^\alpha)_{\alpha<\varepsilon_0}$ is then defined by

$$h^0(x) \stackrel{\text{def}}{=} x, \qquad h^{\alpha+1}(x) \stackrel{\text{def}}{=} h^\alpha(h(x)), \qquad h^\lambda(x) \stackrel{\text{def}}{=} h^{\lambda_x}(x)\,. \tag{34}$$

A closely related hierarchy is the *length hierarchy* $(h_\alpha)_{\alpha<\varepsilon_0}$ defined by

$$h_0(x) \stackrel{\text{def}}{=} 0, \qquad h_{\alpha+1}(x) \stackrel{\text{def}}{=} 1 + h_\alpha(h(x)), \qquad h_\lambda(x) \stackrel{\text{def}}{=} h_{\lambda_x}(x)\,. \tag{35}$$

Last of all, the *fast growing hierarchy* $(f_\alpha)_{\alpha<\varepsilon_0}$ is defined through

$$f_0(x) \stackrel{\text{def}}{=} h(x), \qquad f_{\alpha+1}(x) \stackrel{\text{def}}{=} f_\alpha^{\omega_x}(x), \qquad f_\lambda \stackrel{\text{def}}{=} f_{\lambda_x}(x)\,. \tag{36}$$

Standard versions of these hierarchies are usually defined by setting h as the successor function, in which case they are denoted H^α, H_α, and F_α resp.

Lemma 5.1. *For all $\alpha \in \mathrm{CNF}(\omega^{\omega^\omega})$ and $x \in \mathbb{N}$,*

1. *$h_\alpha(x) = 1 + h_{P_x(\alpha)}(h(x))$ when $\alpha > 0$,*
2. *$h_\alpha(x) \leq h^\alpha(x) - x$,*
3. *$h^{\omega^\alpha \cdot r}(x) = f_\alpha^r(x)$ for all $r < \omega$,*
4. *if h is eventually bounded by F_γ, then f_α is eventually bounded by $F_{\gamma+\alpha}$.*

Bounding the Length Function. Item 1 of Lem. 5.1 shows that M_α and h_α have rather similar expressions, based on derivatives for M_α and predecessors for h_α; they are in fact closely related:

Proposition 5.2. *For all α in $\mathrm{CNF}(\omega^{\omega^\omega})$, there is a constant k s.t. for all $n > 0$, $M_{\alpha,g}(n) \leq h_\alpha(kn)$ where $h(x) \stackrel{\text{def}}{=} x \cdot g(x)$.*

Proposition 5.2 translates for $n, p > 0$ into an

$$L_{\Gamma_p^*,g}(n) \leq h_{\omega^{\omega^{p-1}}}((p-1)n) \quad \text{for } h(x) \stackrel{\text{def}}{=} x \cdot g(x) \tag{37}$$

upper bound on bad (g,n)-controlled sequences in Γ_p^*. We believe (37) answers a wish expressed by Cichoń and Tahhan Bittar in their conclusion [7]: "an appropriate bound should be given by the function $h_{\omega^{\omega^{p-1}}}$, for some reasonable h."

It remains to translate the bound of Prop. 5.2 into a more intuitive and readily usable one. Combined with items 2–4 of Lem. 5.1, Prop. 5.2 allows us to state a fairly general result in terms of the $(\mathscr{F}_\alpha)_\alpha$ classes in the two most relevant cases (of which both the Length Function Theorem given in the introduction and, if $\gamma \geq 2$, the $\mathscr{F}_{\gamma+k}$ bound given for $\mathbb{N}^k$ in [9], are consequences):

Theorem 5.3 (Main Theorem). *Let g be a smooth control function eventually bounded by a function in $\mathscr{F}_\gamma$, and let A be an exponential nwqo with maximal order type $< \omega^{\beta+1}$. Then $L_{A,g}$ is bounded by a function in*

- *$\mathscr{F}_\beta$ if $\gamma < \omega$ (e.g. if g is primitive-recursive) and $\beta \geq \omega$,*
- *$\mathscr{F}_{\gamma+\beta}$ if $\gamma \geq 2$ and $\beta < \omega$.*

6 Refined Complexity Bounds for Verification Problems

This section provides two *examples* where our Main Theorem leads to precise multiply-recursive complexity upper bounds for problems that were known to be decidable but not primitive-recursive. Our choice of examples is guided by our close familiarity with these problems (in fact, they have been our initial motivation for looking at subrecursive hierarchies) and by their current role as master problems for showing Ackermann complexity lower bounds in several areas of verification. (A more explicit vademecum for potential users of the Main Theorem can be found in [9].)

Lossy Channel Systems. The wqo associated with a lossy channel system $S = (Q, M, C, \Delta)$ is the set $A_S \stackrel{\text{def}}{=} Q \times (M^*)^C$ of its configurations, ordered with embedding (see details in [4]). Here Q is a set of q control locations, M is a size-m message alphabet and C is a set of c channels. Hence, we obtain $A_S \equiv q \cdot \left(\Gamma_m^*\right)^c$. For such lossy systems [17], reachability, safety and termination can be decided by algorithms that only need to explore bad sequences over A_S. In particular, S has a non-terminating run from configuration s_{init} iff it has a run of length

$L_{A_S}(|s_{\text{init}}|)$, and the shortest run (if one exists) reaching s_{final} from s_{init} has length at most $L_{A_S}(|s_{\text{final}}|)$. Here the sequences (runs of S, forward or backward) are controlled with $g = Succ$. Now, since $o(A_S) = \omega^{(\omega^{m-1}\cdot c)} \cdot q$, Thm. 5.3 gives an overall complexity at level $\mathscr{F}_{\omega^{(m-1)}\cdot c}$, which is the most precise upper bound so far for lossy channel systems.

Regarding lower bounds, the construction in [4] proves a $\mathscr{F}_{\omega^K}$ lower bound for systems using $m = K+2$ different symbols, $c = 2$ channels, and a quadratic $q \in O(K^2)$ number of states. If emptiness tests are allowed (an harmless extension for lossy systems, see [17]) one can even get rid of the # separator symbol in that construction (using more channels instead) and we end up with $m = K+1$ and $c = 4$. Thus the demonstrated upper and lower bounds are very close, and tight when considering the recursivity-multiplicity level.

$\mathsf{PEP}^{\text{reg}}$, *the Regular Post Embedding Problem,* is an abstract problem that relaxes Post's Correspondence Problem by replacing the equality "$u_{i_1} \dots u_{i_n} = v_{i_1} \dots v_{i_n}$" with embedding "$u_{i_1} \dots u_{i_n} \leq_{\Gamma^*} v_{i_1} \dots v_{i_n}$" (all this under a "$\exists i_1, \dots, i_n$ in some regular R" quantification). It was introduced in [3] where decidability was shown thanks to Higman's Lemma. Non-trivial reductions between $\mathsf{PEP}^{\text{reg}}$ and lossy channel systems exist. Due to its abstract nature, $\mathsf{PEP}^{\text{reg}}$ is a potentially interesting master problem for proving hardness at multiply-recursive and hyper-Ackermannian, i.e., $\mathscr{F}_{\omega^\omega}$, levels (see refs in [5]).

A pumping lemma was proven in [5], which relies on the L_A function, and from which we can now derive more precise complexity upper bounds. Precisely, the proof of Lem. 7.3 in [5] shows that if a $\mathsf{PEP}^{\text{reg}}$ instance admits a solution $\sigma = i_1 \dots i_n$ longer than some bound H then that solution is not the shortest. Here H is defined as $2 \cdot L_{(\Gamma^* \cdot n)}(0)$ for a n that is at most exponential in the size of the instance. Since the control function is linear, Thm. 5.3 yields an $\mathscr{F}_{\omega^{p-1}}$ complexity upper bound for $\mathsf{PEP}^{\text{reg}}$ on a p-letter alphabet (and a hyper-Ackermannian $\mathscr{F}_{\omega^\omega}$ when the alphabet is not fixed). This motivates a closer consideration of lower bounds (left as future work, e.g., by adapting [4]).

7 Concluding Remarks

We proved a general version of the Main Theorem promised in the introduction. Our proof relies on two main components: an algebraic framework for normed wqo's and normed reflections on the one hand, leading on the other hand to descending relations between ordinals that can be captured in subrecursive hierarchies. This setting accommodates all "exponential" wqo's, i.e., finite combinations of Γ_p^*'s. This lets us derive upper bounds for controlled bad sequences when using Higman's Lemma on finite alphabets.

We hope that our framework will extend smoothly beyond exponential wqo's and may also accept additional wqo constructions like powersets, multisets, and perhaps trees.

References

1. Abdulla, P.A., Čerāns, K., Jonsson, B., Tsay, Y.K.: Algorithmic analysis of programs with well quasi-ordered domains. Inform. and Comput. 160, 109–127 (2000)
2. Buchholz, W., Cichoń, E.A., Weiermann, A.: A uniform approach to fundamental sequences and hierarchies. Math. Logic Quart. 40, 273–286 (1994)
3. Chambart, P., Schnoebelen, P.: Post embedding problem is not primitive recursive, with applications to channel systems. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 265–276. Springer, Heidelberg (2007)
4. Chambart, P., Schnoebelen, P.: The ordinal recursive complexity of lossy channel systems. In: Proc. LICS 2008, pp. 205–216. IEEE, Los Alamitos (2008)
5. Chambart, P., Schnoebelen, P.: Pumping and counting on the Regular Post Embedding Problem. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 64–75. Springer, Heidelberg (2010)
6. Cichoń, E.A.: Ordinal complexity measures. In: Conference on Proofs and Computations in Honour of Stan Wainer on the Occasion of his 65th Birthday (2009)
7. Cichoń, E.A., Tahhan Bittar, E.: Ordinal recursive bounds for Higman's Theorem. Theor. Comput. Sci. 201, 63–84 (1998)
8. Clote, P.: On the finite containment problem for Petri nets. Theor. Comput. Sci. 43, 99–105 (1986)
9. Figueira, D., Figueira, S., Schmitz, S., Schnoebelen, P.: Ackermannian and primitive-recursive bounds with Dickson's Lemma. In: Proc. LICS 2011. IEEE, Los Alamitos (to appear, 2011); arXiv:1007.2989 (cs.LO), http://arxiv.org/abs/1007.2989
10. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theor. Comput. Sci. 256, 63–92 (2001)
11. Henzinger, T.A., Majumdar, R., Raskin, J.F.: A classification of symbolic transition systems. ACM Trans. Comput. Logic 6, 1–32 (2005)
12. de Jongh, D.H.J., Parikh, R.: Well-partial orderings and hierarchies. Indag. Math. 39, 195–207 (1977)
13. Kruskal, J.B.: The theory of well-quasi-ordering: A frequently discovered concept. J. Comb. Theory A 13, 297–305 (1972)
14. Löb, M., Wainer, S.: Hierarchies of number theoretic functions, I. Arch. Math. Logic 13, 39–51 (1970)
15. McAloon, K.: Petri nets and large finite sets. Theor. Comput. Sci. 32, 173–183 (1984)
16. Moser, G., Weiermann, A.: Relating derivation lengths with the slow-growing hierarchy directly. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, pp. 296–310. Springer, Heidelberg (2003)
17. Schnoebelen, P.: Lossy counter machines decidability cheat sheet. In: Kučera, A., Potapov, I. (eds.) RP 2010. LNCS, vol. 6227, pp. 51–75. Springer, Heidelberg (2010)
18. Touzet, H.: Propriétés combinatoires pour la terminaison de systèmes des réécriture. Thèse de doctorat, Université de Nancy 1, France (September 1997)
19. Touzet, H.: A characterisation of multiply recursive functions with Higman's Lemma. Inform. and Comput. 178, 534–544 (2002)
20. Weiermann, A.: Complexity bounds for some finite forms of Kruskal's Theorem. J. Symb. Comput. 18, 463–488 (1994)

Liveness-Preserving Atomicity Abstraction*

Alexey Gotsman[1] and Hongseok Yang[2]

[1] IMDEA Software Institute
[2] University of Oxford

Abstract. Modern concurrent algorithms are usually encapsulated in libraries, and complex algorithms are often constructed using libraries of simpler ones. We present the first theorem that allows harnessing this structure to give compositional liveness proofs to concurrent algorithms and their clients. We show that, while proving a liveness property of a client using a concurrent library, we can soundly replace the library by another one related to the original library by a generalisation of a well-known notion of linearizability. We apply this result to show formally that lock-freedom, an often-used liveness property of non-blocking algorithms, is compositional for linearizable libraries, and provide an example illustrating our proof technique.

1 Introduction

Concurrent systems are usually expected to satisfy *liveness* properties [1], which, informally, guarantee that certain good events eventually happen. Reasoning about liveness in modern concurrent programs is difficult. Fortunately, the task can be simplified using reasoning methods that are able to exploit program structure. For example, concurrent algorithms are usually encapsulated in libraries and complex algorithms are often constructed using libraries of simpler ones. Thus, in reasoning about liveness of client code of a concurrent library, we would like to abstract from the details of a particular library implementation. This requires a notion of library abstraction that is able to specify the relevant liveness properties of the library.

Sound abstractions of concurrent libraries are commonly formalised by the notion of *linearizability* [11], which fixes a certain correspondence between the library and its abstract specification (the latter usually sequential, with its methods implemented atomically). However, linearizability is not suitable for liveness. It takes into account finite computations only and does not restrict the termination behaviour of library methods when relating them to methods of an abstract specification. As a result, the linearizing specification loses most of the liveness properties of the library. For example, no linearizing specifications can specify that library methods always terminate or a method meant to acquire a resource may not always return a value signifying that the resource is busy. In this paper we propose a generalisation of linearizability that lifts the above limitations and allows specifying such properties (§3).

Building on our generalized notion of linearizability, we present a theorem that allows giving liveness proofs to concurrent algorithms that are compositional in their structure (Theorem 4, §4). Namely, we show that, while proving a liveness property of

* We would like to thank Anindya Banerjee, Aleks Nanevski, Matthew Parkinson and Viktor Vafeiadis for helpful comments and suggestions. Yang was supported by EPSRC.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 453–465, 2011.

a client of a concurrent library, we can soundly replace the library by a simpler one related to the original library by our generalisation of linearizability. To our knowledge, this is the first result, both for safety and liveness, that allows exploiting linearizability in verifying concurrent programs. In particular, it enables liveness-preserving *atomicity abstraction*. When proving a liveness property of a client using a concurrent library, we can replace the library by its atomic abstract specification, and prove the liveness of the client with respect to this specification instead.

We further show that we can use existing tools for proving classical linearizability (e.g., [16,2]) to establish our generalisation. To this end, we identify a class of *linearization-closed* properties that, when satisfied by a library, are also satisfied by any other library linearizing it. This allows us to perform atomicity abstraction in two stages. We first use existing tools to establish the linearizability of a library to a coarse specification, sufficient only for proving safety properties of a client. We can then strengthen the specification for free with any linearization-closed liveness properties proved of the library implementation (Corollary 7, §5).

Finally, we demonstrate how our results can be used to give compositional proofs of *lock-freedom*, a liveness property often required of modern concurrent algorithms. In particular, we show that lock-freedom is a compositional property for linearizable libraries (Theorem 9, §6): when proving it of an algorithm using a linearizable lock-free library, we can replace library methods by their atomic always-terminating specifications. Our formalisation also highlights a (perhaps surprising) fact that compositionality does not hold for a variant of lock-freedom that assumes fair scheduling. We demonstrate the resulting proof technique on a non-blocking stack by Hendler et al. [9] (§7).

For space reasons, proofs are omitted in the paper. They can be found in [7].

2 Preliminaries

Programming language. We consider a simple language for heap-manipulating concurrent programs, where a program consists of a library implementing several methods and its client, given by a parallel composition of threads. Let Var be a set of variables, ranged over by $x, y, \ldots$, and Method a set of method names, ranged over by m. For simplicity, all methods take one argument. The syntax of the language is given below:

$$\begin{array}{lll} E & ::= \mathbb{Z} \mid x \mid E + E \mid -E \mid \ldots & C ::= c \mid m(E) \mid C; C \mid C + C \mid C^{\circlearrowleft} \\ D & ::= C; \mathsf{return}(E) & (\text{where } C \text{ does not include method calls } m(E)) \\ A & ::= \{m = D, \ldots, m = D\} & \\ C(A) & ::= \mathsf{let}\ A\ \mathsf{in}\ C \parallel \ldots \parallel C & (\text{where all the methods called are defined in } A) \end{array}$$

The language includes primitive commands c, method call $m(E)$, sequential composition $C; C'$, nondeterministic choice $C+C'$, and iteration $C^{\circlearrowleft}$. We use a notation $C^{\circlearrowleft}$ here (instead of the usual Kleene star C^*), so as to emphasise that $C^{\circlearrowleft}$ describes not just finite, but also infinite iterations of C. Both primitive commands and method calls $m(E)$ are assumed atomic. The primitive commands form the set PComm, and they include standard instructions, such as skip, variable assignment $x{=}E$, the update $[E]{=}E'$ of the heap cell E by E', the read $x{=}[E]$ of the heap cell E into the variable x, and the assume statement $\mathsf{assume}(E)$, filtering out states making $E{=}0$. We point out that the standard constructs, such as loops and conditionals, can be defined in our language as syntactic sugar, with conditions translated using assume (see also [7, §A]).

Let us fix a program $C(A) = \mathsf{let}\ A\ \mathsf{in}\ C_1 \parallel \ldots \parallel C_n$ with the library definition $A = \{m = D_m \mid m \in M\}$. We let the signature of the library A be the set of the implemented methods: $\mathsf{sig}(A) = M$. In the following we index threads in programs using the set of identifiers $\mathsf{ThreadID} = \mathbb{N}$.

We restrict programs to ensure that the state of the client is disjoint from that of the library, and that this state separation is respected by client operations and library routines. We note that this restriction is assumed by the standard notion of linearizability [11]; we intend to relax it in our future work. Technically, we assume $\mathsf{PComm} = \mathsf{ClientPComm} \uplus \mathsf{LibPComm}$ for some sets ClientPComm and LibPComm and require that all primitive commands in the client be from ClientPComm and those in the library from LibPComm. As formalised below, the commands in ClientPComm and LibPComm can access only variables and heap locations belonging to the client and the library, respectively.

We also assume special variables $\mathsf{retval}_t \in \mathsf{Var}$ for each thread t and $\mathsf{param} \in \mathsf{Var}$. The variable retval_t contains the result of the most recent method call by thread t and is implicitly updated by the return command. No commands in the program are allowed to modify retval_t explicitly, and the variable can only be read by C_t. The variable param keeps the values of parameters passed upon method calls and is implicitly updated by the call command. We assume that the variables occurring in the expression E of a call command $m(E)$ are accessed only by the thread executing the command.

State model. Let CLoc and LLoc be disjoint sets representing heap locations that belong to the address spaces of the client and the library, respectively. We also assume $\mathsf{Var} = \mathsf{CVar} \uplus \mathsf{LVar}$ for some sets CVar and LVar representing variables that belong to the client and the library, respectively. Let $\mathsf{param} \in \mathsf{LVar}$ and $\mathsf{retval}_t \in \mathsf{CVar}$, $t = 1..n$. We then define the set of program states State as follows:

$$\mathsf{Loc} = \mathsf{CLoc} \uplus \mathsf{LLoc} \qquad \mathsf{Val} = \mathbb{Z} \uplus \mathsf{Loc}$$
$$\mathsf{Stack} = \mathsf{Var} \to \mathsf{Val} \qquad \mathsf{Heap} = \mathsf{Loc} \to \mathsf{Val} \qquad \mathsf{State} = \mathsf{Stack} \times \mathsf{Heap}$$

A state in this model consists of a stack and a heap, both of which are total maps from variables or locations to values. Every location or variable is owned either by the library or by the client. We make this ownership explicit by defining two sets, CState for client states and LState for library states: $\mathsf{CState} = (\mathsf{CVar} \to \mathsf{Val}) \times (\mathsf{CLoc} \to \mathsf{Val})$ and $\mathsf{LState} = (\mathsf{LVar} \to \mathsf{Val}) \times (\mathsf{LLoc} \to \mathsf{Val})$. Also, we define three operations relating these sets to State: $\mathsf{client} : \mathsf{State} \to \mathsf{CState}$, $\mathsf{lib} : \mathsf{State} \to \mathsf{LState}$ and $\circ : \mathsf{CState} \times \mathsf{LState} \to \mathsf{State}$. The first two project states to client and library states by restricting the domains of the stack and the heap: e.g., for $(s, h) \in \mathsf{State}$ we have $\mathsf{lib}(s, h) = (s|_{\mathsf{LVar}}, h|_{\mathsf{LLoc}})$. The $\circ$ operator combines client and library states into program states: $(s_1, h_1) \circ (s_2, h_2) = (s_1 \uplus s_2, h_1 \uplus h_2)$. We lift $\circ$ to sets of states pointwise.

Control-flow graphs. In the definition of program semantics, it is technically convenient for us to abstract from a particular syntax of programming language and represent commands by their ***control-flow graphs***. A control-flow graph (CFG) is a tuple $(N, T, \mathsf{start}, \mathsf{end})$, consisting of the set of program positions N, the control-flow relation $T \subseteq N \times \mathsf{Comm} \times N$, and the initial and final positions $\mathsf{start}, \mathsf{end} \in N$. The edges are annotated with commands from Comm, which are primitive commands, calls $m(E)$ or returns $\mathsf{return}(E)$. Every command C in our language can be translated to a CFG in a standard manner [7, §A]. Hence, we can represent a program $C(A)$ by a collection of CFGs: the client command C_t for a thread t is represented by $(N_t, T_t, \mathsf{start}_t, \mathsf{end}_t)$, and

the body D_m of a method m by $(N_m, T_m, \mathsf{start}_m, \mathsf{end}_m)$. We often view this collection of CFGs for $C(A)$ as a single graph consisting of two node sets $\mathsf{CNode} = \biguplus_{t=1}^{n} N_t$ and $\mathsf{LNode} = \biguplus_{m \in \mathsf{sig}(A)} N_m$, and the edge set $T = \biguplus_{t=1}^{n} T_t \uplus \biguplus_{m \in \mathsf{sig}(A)} T_m$. Finally, we define method : $\mathsf{LNode} \to M$ as follows: $\mathsf{method}(v) = m$ if and only if $v \in N_m$.

Program semantics. Programs in our semantics denote sets of ***traces***, which are finite or infinite sequences of actions of the form

$$\varphi \in \mathsf{Act} \ ::= \ (t, \mathsf{Client}(c)) \mid (t, \mathsf{Lib}(c)) \mid (t, \mathsf{call}\ m(k)) \mid (t, \mathsf{ret}\ m(k))$$

where $t \in \mathsf{ThreadID}$, $c \in \mathsf{PComm}$, $k \in \mathsf{Val}$ and $m \in \mathsf{Method}$. Each action corresponds to a primitive command c executed by the client or the library (in which case we tag c with Client or Lib), a call to or a return from the library. We denote the sets of each kind of actions with ClientAct, LibAct, CallAct and RetAct, respectively, and let $\mathsf{CallRetAct} = \mathsf{CallAct} \cup \mathsf{RetAct}$. Also, we write Trace for the set of all traces and adopt the standard notation: ε is the empty trace, $\tau(i)$ is the i-th action in the trace τ, and $|\tau|$ is the length of the trace τ ($|\tau| = \omega$ if τ is infinite).

Our semantics assumes an interpretation of every primitive command c as a transformer f_c of type $\mathsf{LState} \to \mathcal{P}(\mathsf{LState})$, if $c \in \mathsf{LibPComm}$, or of type $\mathsf{CState} \to \mathcal{P}(\mathsf{CState})$, if $c \in \mathsf{ClientPComm}$. In both cases, we lift it to a function $f_c : \mathsf{State} \to \mathcal{P}(\mathsf{State})$ by transforming only the client or the library part of the input state:

$$\forall \theta \in \mathsf{State}.\ f_c(\theta) \stackrel{\text{def}}{=} \begin{cases} \{\mathsf{client}(\theta)\} \circ f_c(\mathsf{lib}(\theta)), & \text{if } c \in \mathsf{LibPComm}; \\ f_c(\mathsf{client}(\theta)) \circ \{\mathsf{lib}(\theta)\}, & \text{if } c \in \mathsf{ClientPComm}. \end{cases}$$

For the transformers of sample primitive commands see [7, §A].

Let the set of ***thread positions*** be defined as follows: $\mathsf{Pos} = \mathsf{CNode} \cup (\mathsf{Val} \times \mathsf{CNode} \times \mathsf{LNode})$. Elements in Pos describe the runtime status of a thread. A node $v \in \mathsf{CNode}$ indicates that the thread is about to execute the client code coming right after the node v in its CFG. A triple $\langle u, v, v' \rangle$ means that the thread is at the program point v' in the code of a library method, u is the current value of the param variable for this method and v the return program point in the client code.

The semantics of the program $C(A)$ with threads $\{1, \ldots, n\}$ is defined using a transition relation $\longrightarrow_{C(A)} : \mathsf{Config} \times \mathsf{Act} \times \mathsf{Config}$, which transforms program configurations $\mathsf{Config} = (\{1, \ldots, n\} \to \mathsf{Pos}) \times \mathsf{State}$. The configurations are pairs of program counters and states, where a program counter defines the position of each thread in the program. The relation is defined by the rules in Figure 1. Note that, upon a method call, the actual parameter and the return point are saved as components in the new program position, and the method starts executing from the corresponding starting node of its CFG. The saved actual parameter is accessed whenever the library code reads the variable param, as modelled in the third rule for the library action. Upon a return, the return point is read from the current program counter, and the return value is written into the retval_t variable for the thread t executing the return command.

Our operational semantics induces a trace interpretation of programs $C(A)$. For a finite trace τ and $\sigma, \sigma' \in \mathsf{Config}$ we write $\sigma \xrightarrow{\tau}{}^{*}_{C(A)} \sigma'$ if there exists a corresponding derivation of τ using $\longrightarrow$. Similarly, for an infinite trace τ and $\sigma \in \mathsf{Config}$ we write $\sigma \xrightarrow{\tau}{}^{\omega}_{C(A)} -$ to mean the existence of infinite τ-labelled computation from σ according to our semantics. Let us denote with pc_0 the initial program counter $[1 : \mathsf{start}_1, \ldots, n : \mathsf{start}_n]$. The trace semantics of $C(A)$ is defined as follows:

$$[\![C(A)]\!] = \{\tau \mid \exists \theta \in \mathsf{State}.\ (\mathsf{pc}_0, \theta) \xrightarrow{\tau}{}^{\omega}_{C(A)} - \vee \exists \sigma \in \mathsf{Config}.\ (\mathsf{pc}_0, \theta) \xrightarrow{\tau}{}^{*}_{C(A)} \sigma\}.$$

$$\frac{(v,c,v') \in T \quad \theta' \in f_c(\theta)}{\mathsf{pc}[t:v],\theta \xrightarrow{(t,\,\mathsf{Client}(c))}_{C(A)} \mathsf{pc}[t:v'],\theta'} \qquad \frac{(v,m(E),v') \in T \quad [\![E]\!]s = u}{\mathsf{pc}[t:v],\theta \xrightarrow{(t,\,\mathsf{call}\ m(u))}_{C(A)} \mathsf{pc}[t:\langle u,v',\mathsf{start}_m\rangle],\theta}$$

$$\frac{(v,c,v') \in T \quad (s',h') \in f_c(s[\mathsf{param}:u],h)}{\mathsf{pc}[t:\langle u,v_0,v\rangle],(s,h) \xrightarrow{(t,\,\mathsf{Lib}(c))}_{C(A)} \mathsf{pc}[t:\langle s'(\mathsf{param}),v_0,v'\rangle],(s',h')}$$

$$\frac{(v,\mathsf{return}(E),v') \in T \quad [\![E]\!](s[\mathsf{param}:u]) = u'}{\mathsf{pc}[t:\langle u,v_0,v\rangle],(s,h) \xrightarrow{(t,\,\mathsf{ret}\ (\mathsf{method}(v))(u'))}_{C(A)} \mathsf{pc}[t:v_0],(s[\mathsf{retval}_t:u'],h)}$$

Fig. 1. Operational semantics of programs. $[\![E]\!]s \in \mathsf{Val}$ is the value of the expression E in the stack s. We denote with $g[x:y]$ the function that has the same value as g everywhere except x, where it has the value y. We remind the reader that T is the control-flow relation of $C(A)$.

Note that $[\![C(A)]\!]$ includes all finite or infinite traces (including non-maximal ones).

Client and library traces. In this paper we consider two special kinds of traces: ***client traces***, which include only actions from $\mathsf{CallRetAct} \cup \mathsf{ClientAct}$, and ***library traces***, which include only actions from $\mathsf{CallRetAct} \cup \mathsf{LibAct}$. Given a trace $\tau \in \mathsf{Trace}$, we can obtain the corresponding client $\mathsf{client}(\tau)$ and library $\mathsf{lib}(\tau)$ traces by projecting τ to the appropriate subsets of actions. We consider two further projections: $\mathsf{visible}(\tau)$ projects τ to actions in $\mathsf{ClientAct}$, and $\tau|_t$ to actions of thread t. We let CTrace be the set of all client traces and LTrace the set of all library traces.

Histories. We record interactions between the client and the library using ***histories***, which are sequences of actions from $\mathsf{CallRetAct} \cup \mathsf{BlockAct}$, where $\mathsf{BlockAct} = \{(t,\mathsf{starve}) \mid t \in \mathsf{ThreadID}\}$. An action (t,starve) means that thread t is suspended by the scheduler and is never scheduled again (is 'starved'). We record starve events because the liveness properties we are dealing with in this paper, such as lock-freedom (§6), need to distinguish between method non-termination due to divergence and the one due to being starved by the scheduler. Let $\mathsf{History}$ be the set of all histories.

Given a trace $\tau \in \mathsf{Trace}$, we can construct a corresponding history $\mathsf{history}(\tau)$ in two steps. First, for every thread t, if the last action of the thread in τ exists and is an action in $\mathsf{CallAct} \cup \mathsf{LibAct}$, we insert (t,starve) right after this action in τ, obtaining a sequence of actions τ'. The history $\mathsf{history}(\tau)$ is then obtained by projecting τ' to actions in $\mathsf{CallRetAct} \cup \mathsf{BlockAct}$.

3 Concurrent Library Semantics and Linearizability

The correctness of concurrent libraries is usually defined using the notion of linearizability [11], which fixes a particular correspondence between the implementation and the specification of a library. We now define an analogue of this notion in our setting.

Definition 1. *The **linearizability relation** is a binary relation $\sqsubseteq$ on histories defined as follows: $H \sqsubseteq H'$ if $\forall t \in \mathsf{ThreadID}.\, H|_t = H'|_t$ and there is a bijection $\pi\colon \{1,\ldots,|H|\} \to \{1,\ldots,|H'|\}$ such that $\forall i.\, H(i) = H'(\pi(i))$ and*

$$\forall i,j.\ (i<j \wedge H(i) \in \mathsf{RetAct} \wedge H(j) \in \mathsf{CallAct}) \Rightarrow \pi(i) < \pi(j).$$

That is, the history H' linearizes the history H when it is a permutation of the latter preserving the order of actions within threads and non-overlapping method invocations.

The duration of a method invocation is defined by the interval from the method call action to the corresponding return action (or to infinity if there is none). Our definition allows H and H' to be infinite, in contrast to the standard notion of linearizability which considers only finite histories (we provide a more detailed comparison below).

To check if one library linearizes another, we need to define the set of histories a library can generate. We do this using the ***most general client*** of the library. As we show in [7, §B, Decomposition Lemma], the client we define here is indeed most general in the sense that its semantics includes all library behaviours generated by any other client in our programming language. Formally, consider a library $A = \{m = D_m \mid m \in M\}$. For a given n, the most general client $\mathsf{MGC}_n(A)$ is the combination of CFGs for the library A and those for the client with n threads that repeatedly invoke library methods in any order and with all possible parameters: the CFG for thread t is $(N_t, T_t, v^t_{\mathsf{mgc}}, v^t_{\mathsf{mgc}})$ with $N_t = \{v^t_{\mathsf{mgc}}\}$ and $T_t = \{(v^t_{\mathsf{mgc}}, m(u), v^t_{\mathsf{mgc}}) \mid m \in \mathsf{sig}(A), u \in \mathsf{Val}\}$. One can understand $\mathsf{MGC}_n(A)$ as "let A in $C^1_{\mathsf{mgc}} \parallel \ldots \parallel C^n_{\mathsf{mgc}}$" where C^t_{mgc} repeatedly makes all possible method calls. Let $N_{\mathsf{mgc}} = \biguplus_{t=1}^{n} N_t$.

We define the denotation of $\mathsf{MGC}_n(A)$ in a ***library-local semantics***, where the program executes only on the library part of state. Namely, we consider a relation $\longrightarrow_{\mathsf{MGC}_n(A)}$: $\mathsf{LConfig} \times \mathsf{Act} \times \mathsf{LConfig}$ transforming configurations $\mathsf{LConfig} = (\{1, \ldots, n\} \to \mathsf{LPos}) \times \mathsf{LState}$, where $\mathsf{LPos} = N_{\mathsf{mgc}} \cup (\mathsf{Val} \times N_{\mathsf{mgc}} \times \mathsf{LNode})$. The relation is defined as in Figure 1, but with the rule for return commands replaced by the one that does not write the return value into retval_t (since this variable is not part of library states in LState):

$$\frac{(v, \mathsf{return}(E), v') \in T \quad [\![E]\!](s[\mathsf{param} : u]) = u'}{\mathsf{pc}[t : \langle u, v_0, v\rangle], (s, h) \xrightarrow{(t,\, \mathsf{ret}\ (\mathsf{method}(v))(u'))}_{\mathsf{MGC}_n(A)} \mathsf{pc}[t : v_0], (s, h)}$$

Let $[\![\mathsf{MGC}_n(A)]\!]_{\mathsf{lib}} \subseteq \mathsf{LTrace}$ be the set of all traces generated by the program $\mathsf{MGC}_n(A)$ in this semantics from any initial state in LState. We then define the set of all possible behaviours of the library A as the set of its traces $[\![A]\!] = \bigcup_{n \geq 1} [\![\mathsf{MGC}_n(A)]\!]_{\mathsf{lib}}$. This lets us lift the notion of linearizability to libraries as follows.

Definition 2. *For libraries A_1 and A_2 with $\mathsf{sig}(A_1) = \mathsf{sig}(A_2)$ we say that A_2 **linearizes** A_1, written $A_1 \sqsubseteq A_2$, if $\forall H_1 \in \mathsf{history}([\![A_1]\!]).\ \exists H_2 \in \mathsf{history}([\![A_2]\!]).\ H_1 \sqsubseteq H_2$.*

Thus, A_2 linearizes A_1 if every behaviour of the latter under the most general client may be reproduced in a linearized form by the former.

It is instructive to compare our definition of linearizability with the classical one [11]. The original definition of linearizability between an implementation A_1 of a concurrent library and its specification A_2 considers only finite histories without starve actions. It assumes that the specification A_2 is sequential, meaning that every method in it is implemented by an atomic terminating command. To deal with non-terminating method calls in A_1 when comparing the sets of histories generated by the two libraries, the original definition *completes* the histories of A_1 before the comparison: for every call action in the history without a corresponding return action, either the action is discarded or a corresponding return action with an arbitrary return value is added in the history. Such an arbitrary completion of pending calls does not allow making any statements about the termination behaviour of the library in its specification. Furthermore, the fact that the definition considers only finite histories makes it impossible to specify

trace-based liveness properties satisfied by the library, e.g., that a method meant to acquire a resource may not always return a value signifying that the resource is busy.

Our definition lifts these limitations in a bid to enable compositional reasoning about liveness properties of concurrent libraries. Definitions 1 and 2 take into account infinite computations and do not require the specification A_2 to be sequential. Thus, they allow method calls in A_2 to diverge and the execution of such methods to overlap with the executions of others. Our definitions require any method divergence in the implementation to be reproducible in the specification. As we show in §5, by restricting the set of histories of the specification with fairness constraints, we can specify trace-based liveness properties. As we discuss in §8, our definition is more flexible than previous attempts at generalising linearizability to deal with method non-termination.

The classical notion of linearizability can also be expressed in our setting. We use this in §5 to harness existing linearizability checkers in reasoning about liveness properties. The linearizability of concurrent libraries according to the classical definition can be established using several logics and tools, e.g., based on separation logic [16,15] or TVLA [2]. These logics and tools reduce showing linearizability to proving an invariant relating the states of the implementation and the sequential specification. They establish the validity of the invariant both on finite and infinite computations. Hence, it can be shown that they also establish linearizability in the sense of Definition 1. More precisely, assume a specification of the effect of every method m in a library A_1 given as a command $c_m; \mathsf{return}(E_m)$ for some $c_m \in \mathsf{PComm}$. Then the tools in [16,15,2] establish $A_1 \sqsubseteq A_2$, where $A_2 = \{m = (\mathsf{skip}^{\circlearrowleft}; c_m; \mathsf{skip}^{\circlearrowleft}; \mathsf{return}(E_m)) \mid m \in \mathsf{sig}(A_1)\}$. It is easy to show that this linearizability relation, when restricted to finite histories, is equivalent to the classical notion of linearizability [11].

Since the classical notion of linearizability does not specify the termination behaviour of the library, methods in the library A_2 above may diverge. The divergence can happen either before the method makes a change to the library state using c_m or after, as modelled by the two $\mathsf{skip}^{\circlearrowleft}$ statements. In fact, both of these cases are exhibited by practical concurrent algorithms. For example, the classical Treiber's stack [14] and its variations [9] do not modify the state in the case of divergence, but Harris's non-blocking linked list [8] does. History completion in the classical definition of linearizability corresponds to divergence at one of the $\mathsf{skip}^{\circlearrowleft}$ statements in our formalisation: discarding a pending call models the divergence at the first one, and completing a pending call with an arbitrary return the divergence at the second. We are able to express the classical notion of linearizability in a uniform way because the specification A_2 above is not sequential. Namely, we allow non-terminating method invocations to overlap with others in the specification and, hence, do not need to complete them.

4 Atomicity Abstraction

We now show how our notion of linearizability can be used to abstract an implementation of a library while reasoning about liveness properties of its client. Namely, we prove that replacing a library used by a client with its linearization leaves all the original client behaviours reproducible modulo the following notion of trace equivalence:

Definition 3. *Client traces* $\tau, \tau' \in \mathsf{CTrace}$ *are* ***equivalent****, written* $\tau \sim \tau'$*, if* $\forall t \in \mathsf{ThreadID}.\, \tau|_t = \tau'|_t$ *and there exists a bijection* $\pi : \{1, \ldots, |\tau|\} \to \{1, \ldots, |\tau'|\}$ *such that* $\forall i.\, \tau(i) = \tau'(\pi(i))$ *and*

$$\forall i, j.\, (i < j \wedge \tau(i), \tau(j) \in \mathsf{ClientAct}) \Rightarrow \pi(i) < \pi(j)$$

Two traces are equivalent if they are permutations of each other preserving the order of actions within threads and all client actions. Note that $\mathsf{visible}(\tau) = \mathsf{visible}(\tau')$ when $\tau \sim \tau'$. Hence, trace equivalence preserves any linear-time temporal property over trace projections to client actions. The following theorem states the desired abstraction result.

Theorem 4 (Abstraction). *Consider* $C(A_1)$ *and* $C(A_2)$ *such that* $A_1 \sqsubseteq A_2$. *Then*
$\forall \tau_1 \in [\![C(A_1)]\!].\, \exists \tau_2 \in [\![C(A_2)]\!].\, \mathsf{client}(\tau_1) \sim \mathsf{client}(\tau_2) \wedge \mathsf{history}(\tau_1) \sqsubseteq \mathsf{history}(\tau_2)$.

Corollary 5. *If* $A_1 \sqsubseteq A_2$, *then* $\mathsf{visible}([\![C(A_1)]\!]) \subseteq \mathsf{visible}([\![C(A_2)]\!])$.

According to Corollary 5, while reasoning about a client $C(A_1)$ of a library A_1, we can soundly replace A_1 with a simpler library A_2 linearizing A_1: if a linear-time liveness property over client actions holds over $C(A_2)$, it will also hold over $C(A_1)$. In practice, we are usually interested in ***atomicity abstraction*** (see, e.g., [12]), a special case of the above transformation when methods in A_2 are implemented using atomic commands. In §6 we apply this technique to proving liveness properties of modern concurrent algorithms. Before this, however, we need to explain how to establish the required linearizability relation $A_1 \sqsubseteq A_2$. This is the subject of the next section.

5 Linearization-Closed Properties

As we explained in §3, existing tools can only prove linearizability relations $A_1 \sqsubseteq A_2$ such that A_2 has the form $\{m = (\mathsf{skip}^{\circlearrowleft}; c_m; \mathsf{skip}^{\circlearrowleft}; \mathsf{return}(E_m)) \mid m \in \mathsf{sig}(A_1)\}$ for some atomic commands c_m and expressions E_m. The specification A_2 of the library A_1 is too coarse to prove a non-trivial liveness property of a client, as it allows methods to diverge and does not permit specifying liveness properties. If the library A_1 satisfies some liveness property (e.g., its method invocations not starved by the scheduler always terminate), we would like to carry it over to A_2 to use in the proof of the client. This is, in fact, possible for a certain class of properties.

Definition 6. *A property* $P \subseteq \mathsf{History}$ *over histories is* ***linearization-closed*** *if* $\forall H, H'.\, (H \in P \wedge H \sqsubseteq H') \Rightarrow H' \in P$.

Intuitively, linearization-closed properties should be formulated in terms of pairs of call and return actions and not in terms of individual actions, as the latter can be rearranged during linearization. For example, the property "methods that are not starved by the scheduler always terminate" is linearization-closed. In contrast, the property "a $(t, \mathsf{ret}\ m(0))$ action is always followed by a $(t', \mathsf{ret}\ m(1))$ action" is not.

Corollary 7. *Let* $P \subseteq \mathsf{History}$ *be linearization-closed,* $A_1 \sqsubseteq A_2$, *and* $\mathsf{history}([\![A_1]\!]) \subseteq P$. *Then* $\mathsf{visible}([\![C(A_1)]\!]) \subseteq \mathsf{visible}([\![C(A_2)]\!] \cap \{\tau \mid \mathsf{history}(\tau) \in P\})$.

This corollary of Theorem 4, improving on Corollary 5, allows us to perform atomicity abstraction in two stages. We start with the coarse refinement $A_1 \sqsubseteq A_2$ established by tools for classical linearizability. If a liveness property P over histories holds of the implementation A_1 and is linearization-closed, the trace set of the specification A_2 can be shrunk by removing those violating P. To convert $C(A_2)$ into a program with the

trace set $[\![C(A_2)]\!] \cap \{\tau \mid \mathsf{history}(\tau) \in P\}$ we can use standard automata-theoretic techniques from model checking: we represent P by an automaton and construct a synchronous product of $C(A_2)$ and the automaton. See [17,4,6] for more details.

6 Compositional Liveness Proofs for Concurrent Algorithms

Lock-freedom. We now illustrate how Corollary 7 can be used to perform compositional proofs of liveness properties of *non-blocking* concurrent algorithms [10]. These complicated algorithms employ synchronisation techniques alternative to the usual lock-based mutual exclusion and typically provide high-performance concurrent implementations of data structures, such as stacks, queues, linked lists and hash tables (see, for example, the `java.util.concurrent` library).

Out of all properties used to formulate progress guarantees for such algorithms, we concentrate on *lock-freedom*, as the one most often used and most difficult to prove. Informally, an algorithm implementing operations on a concurrent data structure is considered lock-free if from any point in a program's execution, some thread is guaranteed to complete its operation. Thus, lock-freedom ensures the absence of livelock, but not starvation. The formal definition is as follows.

Definition 8. *A library A is **lock-free** if for any $t \in \mathsf{ThreadID}$, the set of its histories* $\mathsf{history}([\![A]\!])$ *satisfies* $\mathsf{LF} = \Box\Diamond(\_, \mathsf{ret}\ \_) \vee \Box((t, \mathsf{call}\ \_) \Rightarrow \Diamond((t, \mathsf{starve}) \vee (t, \mathsf{ret}\ \_)))$.

Here we use linear temporal logic (LTL) over histories, with predicates over actions as atomic propositions; $\Box$ and $\Diamond$ are the standard operators "always" and "eventually" and $\_$ stands for an irrelevant existentially quantified value. The property formalises the informal condition that some operation always complete by requiring that either some operation return infinitely often (for the case when the client calls infinitely many operations), or every operation that has not been starved by the scheduler return (for the case when the client calls only finitely many operations).

Note that the semantics of §2 allows for unfair schedulers that starve some threads. A crucial requirement in the definition of lock-freedom is that the property has to be satisfied under such schedulers: the threads that do get scheduled have to make progress even if others are starved. In Definition 8 we formalise it using `starve` actions.

Example. Consider the algorithm in Figure 2, ignoring the `elim` function and calls to it for now. For readability, the example is presented in C, rather than in our minimalistic language. It is a simple non-blocking implementation of a concurrent stack due to Treiber [14]. A client using the implementation can call several `push` or `pop` operations concurrently. To ensure the correctness of the algorithm, we assume that `pop` does not reclaim the memory taken by the deleted node [10]. The stack is stored as a linked list, and is updated by compare-and-swap (CAS) instructions. CAS takes three arguments: a memory address `addr`, an expected value `v1`, and a new value `v2`. It atomically reads the memory address and updates it with the new value when the address contains the expected value; otherwise, it does nothing. In C syntax this might be written as follows:

```
atomic { if (*addr==v1) {*addr=v2; return 1;} else {return 0;} }
```

In most architectures an efficient CAS (or an equivalent operation) is provided natively by the processor. The operations on the stack are implemented as follows. The function

```
struct Node {
  value_t data;
  Node *next;
};
Node *S;
int collision[SIZE];

void init() { S = NULL; }
void push(value_t v) {
  Node *t, *x;
  x = new Node();
  x->data = v;
  while (1) {
    t = S; x->next = t;
    if (CAS(&S,t,x)) return;
    elim(); } }
```

```
void elim() { // Elimination scheme
  // ...
  int pos = GetPos(); // 0 ≤ pos ≤ SIZE-1
  int hisId = collision[pos];
  while (!CAS(&collision[pos],hisId,MYID))
    hisId = collision[pos];
  // ...
}
value_t pop() {
  Node *t, *x;
  while (1) {
    t = S;
    if (t == NULL) return EMPTY;
    x = t->next;
    if (CAS(&S,t,x)) return t->data;
    elim(); } }
```

Fig. 2. A non-blocking stack implementation

`init` initialises the data structure. The `push` operation *(i)* allocates a new node `x`; *(ii)* reads the current value of the top-of-the-stack pointer `S`; *(iii)* makes the `next` field of the newly created node point to the read value of `S`; and *(iv)* atomically updates the top-of-the-stack pointer with the new value `x`. If the pointer has changed between *(ii)* and *(iv)* and has not been restored to its initial value, the CAS fails and the operation is restarted. The `pop` operation is implemented in a similar way.

Note that a `push` or `pop` operation of Treiber's stack may diverge if other threads are continually modifying `S`: in this case the CAS instruction may always fail, which will cause the operation to restart continually. However, the algorithm is lock-free: if `push` and `pop` execute concurrently, some operation will always terminate.

Lock-freedom of some algorithms, including Treiber's stack, can be proved automatically [6].

Compositionality of lock-freedom. It is easy to check that the property LF in Definition 8 is linearization-closed. Thus, if A is a lock-free library and is linearized by a specification $\{m = (\mathsf{skip}^{\circlearrowleft}; c_m; \mathsf{skip}^{\circlearrowleft}; \mathsf{return}(E_m)) \mid m \in \mathsf{sig}(A)\}$, proving a liveness property of a client of A can be simplified if we replace A by the specification and consider only traces τ of the client such that $\mathsf{history}(\tau)$ satisfies LF. As we now show, in the case when the client is itself a concurrent algorithm being proved lock-free, we can strengthen this result: we can assume a specification of A where every method terminates in all cases. We thus prove that lock-freedom is a compositional property of linearizable libraries.

To simplify presentation, we have not considered nested libraries, which makes the direct formulation of this result impossible. Instead, we rely on the reduction in [6], which says that an algorithm is lock-free if and only if any number of its operations running in parallel do not have infinite traces, i.e., terminate if not starved. Thus, it suffices to prove the result for the termination of the client.

Theorem 9. *Let M be a set of method names. Consider libraries*

$$A_1 = \{m = D_m \mid m \in M\}, \quad A_2 = \{m = (c_m; \mathsf{return}(E_m)) \mid m \in M\},$$
$$A_3 = \{m = (\mathsf{skip}^{\circlearrowleft}; c_m; \mathsf{skip}^{\circlearrowleft}; \mathsf{return}(E_m)) \mid m \in M\},$$

where c_m are atomic commands, A_1 is lock-free, and $A_1 \sqsubseteq A_3$. If $[\![C(A_2)]\!]$ does not have infinite traces, then neither does $[\![C(A_1)]\!]$.

Given the above reduction, the theorem implies that a concurrent algorithm using A_1 is lock-free if it is lock-free when it uses A_2 instead.

We note that some concurrent algorithms rely on locks, while satisfying lock-freedom under the assumption that the scheduler is fair [10]. Perhaps surprisingly, Theorem 9 does not hold in this case: lock-freedom under fair scheduling is not a compositional property. The intuitive reason is as follows: we can replace A_3 with A_2 in Theorem 9 because the effect of operations of A_3 diverging at one of the $\mathsf{skip}^{\circlearrowleft}$ statements is already covered by the possible unfairness of the scheduler. This reasoning becomes invalid once the scheduler is fair. We provide a counterexample in [7, §C].

7 Example

Theorem 9 allows giving proofs of lock-freedom to non-blocking concurrent algorithms that are compositional in the structure of the algorithms, as we now illustrate. As an example, we consider an improvement of Treiber's stack proposed by Hendler, Shavit, and Yerushalmi (HSY), which performs better in the case of higher contention among threads [9]. Figure 2 shows an adapted and abridged version of the algorithm. The implementation combines Treiber's stack with a so-called elimination scheme, implemented by the function `elim` (partially elided). A `push` or a `pop` operation first tries to modify the stack as in Treiber's algorithm, by doing a CAS to change the shared top-of-the-stack pointer. If the CAS succeeds, the operation terminates. If the CAS fails (because of interference from another thread), the operation backs off to the elimination scheme. If this scheme fails, the whole operation is restarted.

The elimination scheme works on data structures that are separate from the list implementing the stack and, hence, can be considered as a library used by the HSY stack with the only method `elim`. The idea behind the scheme is that two contending `push` and `pop` operations can eliminate each other without modifying the stack if `pop` returns the value that `push` is trying to insert. An operation determines the existence of another operation it could eliminate itself with by selecting a random slot `pos` in the `collision` array, and atomically reading that slot and overwriting it with its thread identifier `MYID`. The algorithm implements the atomic read-and-write operation on the `collision` array in a lock-free fashion using CAS. The identifier of another thread read from the array can be subsequently used to perform elimination. The corresponding code does not affect the lock-freedom of the algorithm and is elided in Figure 2.

According to Theorem 9, to prove the lock-freedom of the HSY stack, it is sufficient to prove *(i)* the lock-freedom of the `push` and `pop` with a call to `elim` replaced by its atomic always-terminating specification; and *(ii)* the lock-freedom and linearizability of the `elim` method. The former is virtually identical to the proof of lock-freedom of Treiber's stack, since `elim` acts on data structures disjoint from those of the stack. Informally, this proof is done as follows (see [6] for a detailed formal proof). It is sufficient to check the termination of the program consisting of a parallel composition of an arbitrary number of threads each executing one `push` or `pop` operation. For such a program, we have two facts. First, no thread executes a successful CAS in `push` or `pop`

infinitely often. This is because once the CAS succeeds, the corresponding while-loop terminates. Second, the while-loop in an operation terminates if no other thread executes a successful CAS in `push` or `pop` infinitely often. This is because the operation does not terminate only when its CAS always fails, which requires the other threads to execute the CASes infinitely often. From these two facts, the termination of each thread follows.

The lock-freedom of the `elim` method can be proved in the same way and its linearizability can be proved using existing methods [15,16]. This completes the proof of lock-freedom of the HSY stack. We have thus decomposed the proof of a complicated non-blocking algorithm with two nested loops into proofs of two simple algorithms.

8 Related Work

Filipović et al. [5] have previously characterised linearizability in terms of observational refinement (technically, their result is similar to our Rearrangement Lemma in [7, §B]). They did not consider infinite computations and treated non-terminating methods approximately; thus, they could not handle liveness properties. Besides, the work of Filipović et al. did not justify any compositional proof methods, as we have done in Theorem 4.

Petrank et al. [13] were the first to observe that lock-freedom is compositional, as we prove in Theorem 9. However, their formulation and 'proof' of this property are presented as a piece of informal text talking about artefacts without a clear semantics. As a consequence, their compositionality theorem misses an important requirement that library methods be linearizable.

Burckhardt et al. [3] have attempted to generalise linearizability on finite histories to the case of non-terminating method calls. However, their definition is too restrictive, as it requires any non-termination in the library implementation to be reproducible in the *sequential* specification of the library. This requirement is not satisfied on infinite traces by common lock-free algorithms, where some methods may diverge while others make progress. Additionally, their definition considers any library where methods may modify the library state before diverging (e.g., [8]) as non-linearizable. We provide a more flexible definition.

Atomicity refinement is a well-known method for formal development of concurrent programs [12], which allows refining an atomic specification to a concurrent implementation. As atomicity refinement and abstractions are duals of each other, our results can also be used in the context of formal program development.

References

1. Alpern, B., Schneider, F.B.: Defining liveness. Inf. Process. Lett. 21(4) (1985)
2. Amit, D., Rinetzky, N., Reps, T., Sagiv, M., Yahav, E.: Comparison under abstraction for verifying linearizability. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 477–490. Springer, Heidelberg (2007)
3. Burckhardt, S., Dern, C., Musuvathi, M., Tan, R.: Line-up: A complete and automatic linearizability checker. In: PLDI (2010)
4. Cook, B., Gotsman, A., Podelski, A., Rybalchenko, A., Vardi, M.Y.: Proving that programs eventually do something good. In: POPL (2007)
5. Filipović, I., O'Hearn, P., Rinetzky, N., Yang, H.: Abstraction for concurrent objects. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 252–266. Springer, Heidelberg (2009)

6. Gotsman, A., Cook, B., Parkinson, M., Vafeiadis, V.: Proving that non-blocking algorithms don't block. In: POPL (2009)
7. Gotsman, A., Yang, H.: Liveness-preserving atomicity abstraction, extended version (2011), www.software.imdea.org/~gotsman
8. Harris, T.: A pragmatic implementation of non-blocking linked-lists. In: Welch, J.L. (ed.) DISC 2001. LNCS, vol. 2180, p. 300. Springer, Heidelberg (2001)
9. Hendler, D., Shavit, N., Yerushalmi, L.: A scalable lock-free stack algorithm. In: SPAA (2004)
10. Herlihy, M., Shavit, N.: The art of multiprocessor programming (2008)
11. Herlihy, M.P., Wing, J.M.: Linearizability: a correctness condition for concurrent objects. TOPLAS 12 (1990)
12. Jones, C.: Splitting atoms safely. TCS 375 (2007)
13. Petrank, E., Musuvathi, M., Steensgaard, B.: Progress guarantee via bounded lock-freedom. In: PLDI (2009)
14. Treiber, R.K.: Systems programming: Coping with parallelism. Technical Report RJ 5118, IBM Almaden Research Center (1986)
15. Vafeiadis, V.: Modular fine-grained concurrency verification. PhD Thesis. Technical Report UCAM-CL-TR-726, University of Cambridge (2008)
16. Vafeiadis, V.: Automatically proving linearizability. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 450–464. Springer, Heidelberg (2010)
17. Vardi, M.Y.: Verification of concurrent programs—the automata-theoretic framework. Annals of Pure and Applied Logic 51 (1991)

On Stabilization in Herman's Algorithm*

Stefan Kiefer[1], Andrzej S. Murawski[2], Joël Ouaknine[1], James Worrell[1], and Lijun Zhang[3]

[1] Department of Computer Science, University of Oxford, UK
[2] Department of Computer Science, University of Leicester, UK
[3] DTU Informatics, Technical University of Denmark, Denmark

Abstract. Herman's algorithm is a synchronous randomized protocol for achieving self-stabilization in a token ring consisting of N processes. The interaction of tokens makes the dynamics of the protocol very difficult to analyze. In this paper we study the expected time to stabilization in terms of the initial configuration.

It is straightforward that the algorithm achieves stabilization almost surely from any initial configuration, and it is known that the worst-case expected time to stabilization (with respect to the initial configuration) is $\Theta(N^2)$. Our first contribution is to give an upper bound of $0.64N^2$ on the expected stabilization time, improving on previous upper bounds and reducing the gap with the best existing lower bound. We also introduce an asynchronous version of the protocol, showing a similar $O(N^2)$ convergence bound in this case.

Assuming that errors arise from the corruption of some number k of bits, where k is fixed independently of the size of the ring, we show that the expected time to stabilization is $O(N)$. This reveals a hitherto unknown and highly desirable property of Herman's algorithm: it recovers quickly from bounded errors. We also show that if the initial configuration arises by resetting each bit independently and uniformly at random, then stabilization is significantly faster than in the worst case.

1 Introduction

Self-stabilization is a concept of fault-tolerance in distributed computing. A system is self-stabilizing if, starting in an arbitrary state, it reaches a correct or legitimate state and remains in a legitimate state thereafter. Thus a self-stabilizing system is able to recover from *transient errors* such as state-corrupting faults. The study of self-stabilizing algorithms originated in an influential paper of Dijkstra [4]. By now there is a considerable body of work in the area, see [18,5].

In this paper we consider self-stabilization in a classical context that was also treated in Dijkstra's original paper—a *token ring*, i.e., a ring of N identical processes, exactly one of which is meant to hold a token at any given time. If, through some error, the ring enters a configuration with multiple tokens, self-stabilization requires that the system be guaranteed to reach a configuration with

* Research supported by EPSRC (EP/G069158/1). Stefan Kiefer is supported by a postdoctoral fellowship of the German Academic Exchange Service (DAAD).

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 466–477, 2011.

only one token. In particular, we are interested in analyzing a self-stabilization algorithm proposed by Herman [11].

Herman's algorithm is a randomized procedure by which a ring of processes connected uni-directionally can achieve self-stabilization almost surely. The algorithm works by having each process synchronously execute the following action at each time step: if the process possesses a token then it passes the token to its clockwise neighbor with probability 1/2 and keeps the token with probability 1/2. If such a process decides to keep its token and if it receives a token from its neighbor then the two tokens are annihilated. Due to the way the algorithm is implemented we can assume that an error state always has an odd number of tokens, thus this process of pairwise annihilation eventually leads to a configuration with a single token.

While the almost-sure termination of Herman's algorithm is straightforward, computing the time to termination is a challenging problem. This is characteristic of systems of interacting particles under random motion, which are ubiquitous in the physical and medical sciences, including statistical mechanics, neural networks and epidemiology [15]. The analysis of such systems typically requires delicate combinatorial arguments [6]. Our case is no exception, and we heavily exploit work of Balding [1], which was motivated by a scenario from physical chemistry.

Given some initial configuration, let $\mathbf{T}$ be the time until the token ring stabilizes under Herman's algorithm. We analyze the expectation of $\mathbf{T}$ in three natural cases: the worst case (over all initial configurations); the case in which the initial configuration is chosen uniformly at random; the case in which the initial configuration arises from a legitimate configuration by a bounded number of bit errors. In addition we introduce and analyze an asynchronous variant of Herman's algorithm. The latter dispenses with the successive time steps required in the synchronous algorithm, and instead has each process pass its token after an exponentially distributed time delay.

Herman's original paper [11] showed that $\mathbb{E}\mathbf{T} \leq (N^2 \log N)/2$ in the worst case (i.e., over all initial configurations with N processes). It also mentions an improved upper bound of $O(N^2)$ due to Dolev, Israeli, and Moran, without giving a proof or a further reference. In 2005, three papers [9,16,17] were published, largely independently, all of them giving improved $O(N^2)$ bounds. The paper [16] also gives a lower bound of $4N^2/27$, which is the expected stabilization time starting from a configuration with three equally spaced tokens. It was conjectured in [16] that this is the worst case among all starting configurations, including those with more than three tokens. This intriguing conjecture is supported by experimental evidence [2].

Our first result, Theorem 2, gives an upper bound of $0.64N^2$ for the expected stabilization time in the synchronous version of Herman's protocol (improving the constant in the hitherto best bound by a third). We also give an upper bound in the asynchronous case. To the best of our knowledge this is the first analysis of an asynchronous version of Herman's algorithm.

To understand the other main results of the paper requires some detail of the implementation of Herman's algorithm. We assume that each process has a bit that it can read and write, and that each process can read the bit of its counterclockwise neighbor. A process's bit does not directly indicate the presence of a token, rather a process has a token if it has the same bit as its counterclockwise neighbor. Token passing is then implemented by having processes flip their bits.

In Theorem 7 we provide an upper bound on the expected time to stabilize starting from the random initial configuration, that is, the configuration in which each process's bit is reset independently and uniformly at random. Herman's algorithm is such that the random configuration is obtained in one step from the *full configuration*, i.e., the configuration in which every process has a token. The upper bound for the random configuration is far better than the worst-case bound in Theorem 2; in particular, there are three-token configurations for which $\mathbb{E}\mathbf{T}$ is provably larger than the upper bound for the random configuration.

In Theorem 8 we show that for configurations that are obtained from a legitimate configuration by flipping a constant number of process bits, we have $\mathbb{E}\mathbf{T} = O(N)$; i.e., the expected *restabilization* time is linear in N. This contrasts with the fact that there are configurations, even with only three tokens, that need $\Omega(N^2)$ expected time for self-stabilization. Intuitively, our result points at a highly desirable—and, to the best of our knowledge, previously unknown—feature of Herman's protocol: it recovers quickly from bounded errors. This is related to the notion of a *time adaptive protocol* from [14], which refers to a protocol whose recovery time depends on the number of state-corrupted nodes rather than the total number of nodes.

Full proofs are given in [13].

Related Work. One parameter in the design of self-stabilizing algorithms is the number of states per machine. In [8], three different self-stabilizing algorithms with two states per machine are investigated. Only one of those algorithms works in a unidirectional ring, the other algorithms need more connections. The ring algorithm is probabilistic, but it is not symmetric: it requires an "exceptional machine" which executes different code. Herman's algorithm is mentioned in [8] as another two-state algorithm, but it is criticized by saying "it requires that all machines make moves synchronously which is not easily done". In this paper, we suggest and analyze an asynchronous variant of Herman's algorithm, which is symmetric and has only two states per machine.

The protocol of [12], also described in [2], is similar to Herman's protocol in that tokens are passed on a ring of processors. A scheduler selects a processor among those with a token; the selected processor passes the token to left or right neighbor, with probability 0.5, respectively. Two colliding tokens are *merged* to a single token. Our analysis of the asynchronous version of Herman's protocol could possibly be adapted to this protocol, by assuming that a processor passes its token after an exponentially distributed holding time. Of course, the fact that meeting tokens are merged and not annihilated would have to be taken into account.

2 Preliminaries

We assume N processors, with N odd, organized in a ring topology. Each processor may or may not have a token. Herman's protocol in the traditional *synchronous variant* [11] works as follows: in each time step, each processor that has a token passes its token to its clockwise neighbor with probability r (where $0 < r < 1$ is a fixed parameter), and keeps it with probability $1-r$; if a processor keeps its token and receives another token from its counterclockwise neighbor, then both of those tokens are annihilated. Notice that the number of tokens never increases, and can decrease only by even numbers.

Herman's protocol can be implemented as follows. Each processor possesses a bit, which the processor can read and write. Each processor can also read the bit of its counterclockwise neighbor. In this representation having the same bit as one's counterclockwise neighbor means having a token. In each time step, each processor compares its bit with the bit of its counterclockwise neighbor; if the bits are different, the processor keeps its bit; if the bits are equal, the processor flips its bit with probability r and keeps it with probability $1 - r$. It is straightforward to verify that this procedure implements Herman's protocol: in particular a processor flipping its bit corresponds to passing its token to its clockwise neighbor.[1]

We denote the number of initial tokens by M, where $1 \leq M \leq N$. The token representation described above enforces that M be odd. A configuration with only one token is called *legitimate*. The protocol can be viewed as a Markov chain with a single bottom SCC in which all states are legitimate configurations. So a legitimate configuration is reached with probability 1, regardless of the initial configuration, that is, the system *self-stabilizes* with probability 1.

In this paper we also propose and analyze an *asynchronous variant* of Herman's protocol which works similarly to the synchronous version. The asynchronous variant gives rise to a continuous-time Markov chain. Each processor with a token passes the token to its clockwise neighbor with rate λ, i.e., a processor keeps its token for a time that is distributed exponentially with parameter λ, before passing the token to its clockwise neighbor (i.e., flipping its bit). The advantage of this variant is that it does not require processor synchronization. Note that a processor can approximate an exponential distribution by a geometric distribution, that is, it can execute a loop which it leaves with a small fixed probability at each iteration. A more precise approximation can be obtained using a random number generator and precise clocks. For our performance analyses we assume an exact exponential distribution.

Let $\mathbf{T}$ denote the time until only one token is left, i.e., until self-stabilization has occurred. In this paper we analyze the random variable $\mathbf{T}$, focusing mainly

[1] Notice that flipping all bits in a given configuration keeps all tokens in place. In fact, in the original formulation [11], in each iteration each bit is effectively flipped once more, so that flipping the bit means keeping the token, and keeping the bit means passing the token. The two formulations are equivalent in the synchronous version, but our formulation allows for an asynchronous version.

on its expectation $\mathbb{E}\mathbf{T}$. Many of our results hold for both the synchronous and the asynchronous protocol version.

To aid our analysis we think of the processors as numbered from 1 to N, clockwise, according to their position in the ring. We write $m := (M-1)/2$. Let $z : \{1,\dots,M\} \to \{1,\dots,N\}$ be such that $z(1) < \cdots < z(M)$ and for all $i \in \{1,\dots,M\}$, the processor $z(i)$ initially has a token; in other words, $z(i)$ is the position of the i-th token. We often write z_{uv} for $z(v) - z(u)$.

3 Bounds on $\mathbb{E}\mathbf{T}$ for Arbitrary Configurations

The following proposition gives a precise formula for $\mathbb{E}\mathbf{T}$ in both the synchronous and asynchronous protocols in case the number of tokens is $M = 3$.

Proposition 1 (cf. [16]). *Let N denote the number of processors and let a, b, c denote the distances between neighboring tokens, so that $a+b+c = N$. For the synchronous protocol with parameter r let $D = r(1-r)$, and for the asynchronous protocol with parameter λ let $D = \lambda$. Then the expected time to stabilization is*

$$\mathbb{E}\mathbf{T} = \frac{abc}{DN}\,.$$

Proposition 1 is shown in [16] for the synchronous case with $r = \frac{1}{2}$. Essentially the same proof works for $0 < r < 1$, and also in the asynchronous case.

We call a configuration with $M = 3$ equally spaced tokens an *equilateral configuration*. If N is an odd multiple of 3 then $a = b = c = N/3$ for the equilateral configuration. If N is not a multiple of 3 then we ask that a, b, c equal either $\lfloor N/3 \rfloor$ or $\lceil N/3 \rceil$. By Proposition 1 the expected stabilization time for a equilateral configuration is $\mathbb{E}\mathbf{T} = \frac{N^2}{27D}$. It follows that for configurations with $M = 3$ the worst case is $\mathbb{E}\mathbf{T} = \Omega(N^2)$ and this case arises for the equilateral configuration. In fact it has been conjectured in [16] that, for all N, the equilateral configuration is the worst case, not only among the configurations with $M = 3$, but among all configurations. This conjecture is supported by experiments carried out using the probabilistic model checker PRISM—see [2].

Finding upper bounds on $\mathbb{E}\mathbf{T}$ in the synchronous case goes back to Herman's original work [11]. He does not analyze $\mathbb{E}\mathbf{T}$ in the journal version, but in his technical report [11], where he proves $\mathbb{E}\mathbf{T} \le N^2\lceil \log N \rceil/2$. He also mentions an improvement to $O(N^2)$ due to Dolev, Israeli, and Moran, without giving a proof or a further reference. In 2005, three papers [9,16,17] were published, largely independently, all of them giving improved $O(N^2)$ bounds. In [9] path-coupling methods are applied to self-stabilizing protocols, which lead in the case of Herman's protocol to the bound $\mathbb{E}\mathbf{T} \le 2N^2$ for the case $r = \frac{1}{2}$. Independently, the authors of [16] claimed $O(N^2)$. Their proof is elementary and also shows $\mathbb{E}\mathbf{T} \le 2N^2$ for the case $r = \frac{1}{2}$. Finally, the author of [17] (being aware of the conference version of [9]) applied the theory of coalescing random walks to Herman's protocol to obtain $\mathbb{E}\mathbf{T} \le \left(\frac{\pi^2}{8} - 1\right) \cdot \frac{N^2}{r(1-r)}$, which is about $0.93N^2$ for

the case $r = \frac{1}{2}$. By combining results from [17] and [16], we further improve the constant in this bound (by 8/27, which is about 32%), and at the same time generalize it to the asynchronous protocol.

Theorem 2. *For the synchronous protocol with parameter r let $D = r(1-r)$, and for the asynchronous protocol with parameter λ let $D = \lambda$. Then, for all N and for all initial configurations, we have*

$$\mathbb{E}\mathbf{T} \leq \left(\frac{\pi^2}{8} - \frac{29}{27}\right) \cdot \frac{N^2}{D}\,.$$

Hence, $\mathbb{E}\mathbf{T} \leq 0.64N^2$ in the synchronous case with $r = \frac{1}{2}$.

4 Expressions for $\mathbb{E}\mathbf{T}$

Our analysis of Herman's protocol exploits the work of Balding [1] on annihilating particle systems. Such systems are a special case of *interacting particle systems*, which model finitely or infinitely many particles, which, in the absence of interaction, would be modeled as independent Markov chains. Due to particle interaction, the evolution of a single particle is no longer Markovian. Interacting particle systems have applications in many fields, including statistical mechanics, neural networks, tumor growth and spread of infections, see [15]. Balding's paper [1] is motivated by a scenario from physical chemistry, where particles can be viewed as vanishing on contact, because once two particles have met, they react and are no longer available for reactions afterwards. We refer the reader to [10] and the references therein for more information on such chemical reaction systems.

We transfer results from [1] to Herman's protocol. The setup is slightly different because, unlike chemical particles, the tokens in Herman's protocol move only in one direction. This difference is inconsequential, as the state of a system can be captured using only relative token (or particle) distances. Care must be taken though, because Balding does not consider "synchronous" particle movement (this would make no sense in chemistry), but particles moving "asynchronously" or continuously in a Brownian motion.

Given two tokens u and v with $1 \leq u < v \leq M$, we define a random variable $\mathbf{T}_{uv}$ and events $A_{(uv)\downarrow}$ and $A_{(uv)\uparrow}$ in terms of a system in which collisions between tokens u and v cause u and v to be annihilated, but the movement of the other tokens and their possible collisions are ignored. In that system, $\mathbf{T}_{uv}$ denotes the time until u and v have collided. Further, let $A_{(uv)\downarrow}$ and $A_{(uv)\uparrow}$ denote the events that tokens u and v eventually collide *down* and *up*, respectively. By colliding down (resp. up) we mean that, upon colliding, the token u (resp. v) has caught up with v (resp. u) in clockwise direction; more formally, if $d_u, d_v \geq 0$ denote the distances travelled in clockwise direction by the tokens until collision, then the collision is said to be down (resp. up) if $z(u) + d_u = z(v) + d(v)$ (resp. $z(u) + d_u + N = z(v) + d_v$). The behavior of two such tokens is equivalent to

that of a one-dimensional random walk on $\{0,\dots,N\}$, started at z_{uv}, with absorbing barriers at 0 and N: the position in the random walk corresponds to the distance between the tokens, and colliding down (resp. up) corresponds to being absorbed at 0 (resp. N). By this equivalence we have $\mathcal{P}\left(A_{(uv)\downarrow}\right) = 1 - z_{uv}/N$ and $\mathcal{P}\left(A_{(uv)\uparrow}\right) = z_{uv}/N$ (see, e.g., [7]).

Proposition 3 below allows to express the distribution of $\mathbf{T}$ in terms of the distribution of $\mathbf{T}_{uv}$, conditioned under $A_{(uv)\downarrow}$ and $A_{(uv)\uparrow}$, respectively. Those distributions are well-known [7,3]. For the statement we need to define the set W_M of all *pairings*. A pairing is a set $w = \{(u_1,v_1),\dots,(u_m,v_m)\}$ with $1 \le u_i < v_i \le M$ for all i, such that there is $w_0 \in \{1,\dots,M\}$ with $\{u_1,v_1,\dots,u_m,v_m,w_0\} = \{1,\dots,M\}$. Define $s(w) = 1$ if the permutation $(u_1v_1\cdots u_mv_mw_0)$ is even, and $s(w) = -1$ otherwise. (This is well-defined: it is easy to see that $s(w)$ does not depend on the order of the (u_i,v_i).) We have the following proposition:

Proposition 3 (cf. [1, Theorem 2.1]). *Let $M \ge 3$. For all $t \ge 0$:*

$$\mathcal{P}\left(\mathbf{T} \le t\right) = \sum_{w \in W_M} s(w) \prod_{(u,v)\in w} \left(\mathcal{P}\left(\mathbf{T}_{uv} \le t \cap A_{(uv)\downarrow}\right) - \mathcal{P}\left(\mathbf{T}_{uv} \le t \cap A_{(uv)\uparrow}\right)\right) .$$

Balding's Theorem 2.1 in [1] is more general in that it gives a generating function for the number of remaining tokens at time t. Strictly speaking, Balding's theorem is not applicable to the synchronous version of Herman's protocol, because he only considers tokens that move according to the asynchronous version (in our terms), and tokens in a Brownian motion. In addition, his proof omits many details, so we give a self-contained proof for Proposition 3 in [13].

Theorem 4 below yields an expression for $\mathbb{E}\mathbf{T}$. We define the set $\overrightarrow{W_M}$ of all *directed pairings* as the set of all sets $\overrightarrow{w} = \{(u_1,v_1,d_1),\dots,(u_m,v_m,d_m)\}$ such that $\{(u_1,v_1),\dots,(u_m,v_m)\} \in W_M$ and $d_i \in \{\downarrow,\uparrow\}$ for all $i \in \{1,\dots,m\}$. For a directed pairing $\overrightarrow{w} = \{(u_1,v_1,d_1),\dots,(u_m,v_m,d_m)\}$ we define

$$\overrightarrow{s}(\overrightarrow{w}) := s(\{(u_1,v_1),\dots,(u_m,v_m)\}) \cdot (-1)^{|\{i \mid 1\le i\le m,\ d_i=\uparrow\}|}$$

and the event $A_{\overrightarrow{w}} := \bigcap_{i=1}^m A_{(u_iv_i)d_i}$. Notice that $\mathcal{P}\left(A_{\overrightarrow{w}}\right) = \prod_{i=1}^m \mathcal{P}\left(A_{(u_iv_i)d_i}\right)$. Further, we set $\mathbf{T}_{\overrightarrow{w}} := \max\{\mathbf{T}_{u_iv_i} \mid 1 \le i \le m\}$. We have the following theorem:

Theorem 4. *For $M \ge 3$:*

$$\mathbb{E}\mathbf{T} = \sum_{\overrightarrow{w} \in \overrightarrow{W_M}} \overrightarrow{s}(\overrightarrow{w}) \cdot \mathbb{E}\left[\mathbf{T}_{\overrightarrow{w}} \mid A_{\overrightarrow{w}}\right] \cdot \mathcal{P}\left(A_{\overrightarrow{w}}\right) .$$

A Finite Expression for $\mathbb{E}\mathbf{T}$. In the rest of the section we focus on the synchronous protocol. We obtain a closed formula for $\mathbb{E}\mathbf{T}$ in Proposition 5 below.

For $1 \le u < v < M$, we define $z_{uv\downarrow} := z_{uv}$ and $z_{uv\uparrow} := N - z_{uv}$. For sets $\emptyset \ne \overrightarrow{x} \subseteq \overrightarrow{w} \in \overrightarrow{W_M}$ with $\overrightarrow{x} = \{(u_1,v_1,d_1),\dots,(u_k,v_k,d_k)\}$ and $\overrightarrow{w} = \{(u_1,v_1,d_1),\dots,(u_m,v_m,d_m)\}$ we write

$$y_F(\overrightarrow{x},\overrightarrow{w}) := \left(\frac{z_{u_1v_1d_1}}{N},\ldots,\frac{z_{u_kv_kd_k}}{N}\right) \quad \text{and}$$
$$y_G(\overrightarrow{x},\overrightarrow{w}) := \left(\frac{z_{u_{k+1}v_{k+1}d_{k+1}}}{N},\ldots,\frac{z_{u_mv_md_m}}{N}\right) .$$

Let

$$g(j,y;u) := \frac{\sin(j\pi y)\cdot\sin(j\pi u)}{1-\cos(j\pi u)} \qquad \text{and} \qquad h(j;u) := 1-2r(1-r)\,(1-\cos(j\pi u)) ,$$

and define, for $k \in \mathbb{N}_+$ and $\ell \in \mathbb{N}_+$,

$$F_k^{(N)}(y_1,\ldots,y_k) := -\left(\frac{-1}{N}\right)^k \cdot \sum_{j\in\{1,\ldots,N-1\}^k} \frac{\prod_{i=1}^k g(j(i),y_i;1/N)}{1-\prod_{i=1}^k h(j(i);1/N)} \qquad \text{and}$$

$$G_\ell(y_1,\ldots,y_\ell) := \prod_{i=1}^{\ell}(1-y_i) .$$

We drop the subscripts of $F_k^{(N)}$ and G_ℓ, if they are understood. Observe that $F^{(N)}$ and G are continuous and do not depend on the order of their arguments. The following proposition gives, for the synchronous protocol, a concrete expression for $\mathbb{E}\mathbf{T}$.

Proposition 5. *Consider the synchronous protocol. For $M \geq 3$:*

$$\mathbb{E}\mathbf{T} = \sum_{\overrightarrow{w}\in\overrightarrow{W_M}} \overrightarrow{s}(\overrightarrow{w}) \sum_{\emptyset\neq\overrightarrow{x}\subseteq\overrightarrow{w}} F^{(N)}(y_F(\overrightarrow{x},\overrightarrow{w}))\cdot G(y_G(\overrightarrow{x},\overrightarrow{w})) .$$

An Approximation for $\mathbb{E}\mathbf{T}$. The function $F^{(N)}$ in Proposition 5 depends on N, and also on r. This prohibits a deeper analysis as needed in Section 6. Proposition 6 gives an approximation of $\mathbb{E}\mathbf{T}$ without those dependencies. To state it, we define, for $k \in \mathbb{N}_+$, a function $\widetilde{F}_k : [0,1]^k \to \mathbb{R}$ with

$$\widetilde{F}_k(y_1,\ldots,y_k) = \frac{-1}{\pi^2}\left(\frac{-2}{\pi}\right)^k \sum_{j\in\mathbb{N}_+^k} \frac{\prod_{i=1}^k \sin(y_i j(i)\pi)}{\left(\prod_{i=1}^k j(i)\right)\left(\sum_{i=1}^k j(i)^2\right)} .$$

We drop the subscript of $\widetilde{F}_k$, if it is understood. It is shown in [13] that the series in $\widetilde{F}_k$ converges. We have the following proposition.

Proposition 6. *Consider the synchronous protocol. Let*

$$\widetilde{E} := \frac{N^2}{r(1-r)} \sum_{\overrightarrow{w}\in\overrightarrow{W_M}} \overrightarrow{s}(\overrightarrow{w}) \sum_{\emptyset\neq\overrightarrow{x}\subseteq\overrightarrow{w}} \widetilde{F}(y_F(\overrightarrow{x},\overrightarrow{w}))\cdot G(y_G(\overrightarrow{x},\overrightarrow{w})) .$$

Then, for each fixed $M \geq 3$ and $r \in \left(\frac{1}{2}-\frac{\sqrt[4]{27}}{6},\frac{1}{2}+\frac{\sqrt[4]{27}}{6}\right) \approx (0.12,0.88)$ and $\varepsilon > 0$,

$$\mathbb{E}\mathbf{T} = \widetilde{E} + O(N^\varepsilon) .$$

The proof of Proposition 6 is elementary but involved.

5 The Full Configuration

In this section we consider the initial configuration in which every processor has a token, i.e., $N = M$. We call this configuration *full*. Notice that in the full configuration, with all bits set to 0, in the successor configuration each bit is independently set to 1 with probability r. Thus we study the full configuration in lieu of the random configuration. We have the following theorem:

Theorem 7. *For the synchronous protocol with parameter r let $D = r(1-r)$. For the asynchronous protocol with parameter $\lambda > 0$ let $D = \lambda$. For almost all odd $N \in \mathbb{N}_+$, we have for the full configuration:*

$$\mathbb{E}\mathbf{T} \leq 0.0285N^2/D \quad \textit{and} \quad \mathcal{P}\left(\mathbf{T} \geq 0.02N^2/D\right) < 0.5\,.$$

Recall from Proposition 1 that, for N an odd multiple of 3, we have $\mathbb{E}\mathbf{T} = \frac{1}{27}\frac{N^2}{D} \approx 0.0370\frac{N^2}{D}$ if we start from the equilateral configuration. It follows that, for large N, the full configuration (with $M = N$) stabilizes faster than the equilateral configuration (with $M = 3$). This is consistent with the aforementioned conjecture of McIver and Morgan that the equilateral configuration with $M = 3$ is the worst case among all configurations for a fixed N.

6 Restabilization

In this section we restrict attention to the synchronous version of Herman's algorithm and consider the standard bit-array implementation. Theorem 2 shows that the worst-case expected time to termination, considering all initial configurations, is $\mathbb{E}\mathbf{T} = O(N^2)$. We imagine that an initial configuration represents the state of the system immediately after an error, that is, the ring of tokens has become illegitimate because some of positions in the bit array were corrupted. In this light a natural restriction on initial configurations is to consider those that arise from a one-token configuration by corrupting some fixed number m of bits. We call these *flip-m* configurations. Notice that, by the token representation in Herman's protocol, a single bit error can lead to the creation of two neighboring tokens. So, m bit errors could lead to the creation of m new pairs of neighboring tokens. It could also happen that two bit errors affect neighboring bits, leading to a new pair of tokens at distance 2. To account for this, we characterize flip-m configuration as those with at most $2m + 1$ tokens such that the tokens can be arranged into pairs, each pair at distance at most m, with one token left over.

Fixing the number of bit errors we show that the expected time to restabilization improves to $O(N)$. Formally we show:

Theorem 8. *Consider the synchronous protocol. Fix any $m \in \mathbb{N}_+$ and $r \in \left(\frac{1}{2} - \frac{\sqrt[4]{27}}{6}, \frac{1}{2} + \frac{\sqrt[4]{27}}{6}\right) \approx (0.12, 0.88)$. Then for any flip-$m$ configuration we have $\mathbb{E}\mathbf{T} = O(N)$.*

Proof. It suffices to consider flip-m configurations with $M = 2m + 1$ tokens. Without loss of generality, we assume that, when removing token $2m + 1$, the token pairs $(1, 2), (3, 4), \ldots, (2m - 1, 2m)$ have distances at most m; i.e., we assume $z(u + 1) - z(u) \leq m$ for all odd u between 1 and $2m - 1$.

For each directed pairing $\overrightarrow{w} \in \overrightarrow{W_M}$, we define its *class* $Cl(\overrightarrow{w})$ and its *companion pairing* $\overrightarrow{w}' \in \overrightarrow{W_M}$. For the following definition, we define $\widetilde{u} := u + 1$, if u is odd, and $\widetilde{u} := u - 1$, if u is even.

- If $(u, M, d) \in \overrightarrow{w}$ for some u, then $Cl(\overrightarrow{w}) = 0$. Its companion pairing is obtained, roughly speaking, by u and $\widetilde{u}$ switching partners. More precisely:
  - If $(\widetilde{u}, v, d')$ (resp. $(v, \widetilde{u}, d')$) for some (v, d'), then the companion pairing of w is obtained by replacing (u, M, d) and $(\widetilde{u}, v, d')$ with $(\widetilde{u}, M, d)$ and (u, v, d') (resp. (v, u, d')).
  - Otherwise (i.e., $\widetilde{u}$ does not have a partner), the companion pairing of w is obtained by replacing (u, M, d) with $(\widetilde{u}, M, d)$.
- If $\overrightarrow{w} = \{(1, 2, d_1), (3, 4, d_2), \ldots, (M - 2, M - 1, d_m)\}$ for some $d_1, \ldots, d_m$, then $Cl(\overrightarrow{w}) = m$. In this case, $\overrightarrow{w}$ does not have a companion pairing.
- Otherwise, $Cl(\overrightarrow{w})$ is the greatest number i such that for all $1 \leq j \leq i - 1$, the tokens $2j - 1$ and $2j$ are partners (i.e., $(2j - 1, 2j, d)$ for some d). Notice that $0 < Cl(\overrightarrow{w}) < m$. The companion pairing of $\overrightarrow{w}$ is obtained by $2i - 1$ and $2i$ switching partners.

It is easy to see that, for any $\overrightarrow{w} \in \overrightarrow{W_M}$ with $Cl(\overrightarrow{w}) < m$, we have $Cl(\overrightarrow{w}) = Cl(\overrightarrow{w}')$, and the companion pairing of $\overrightarrow{w}'$ is $\overrightarrow{w}$, and $\overrightarrow{s}(\overrightarrow{w}) = -\overrightarrow{s}(\overrightarrow{w}')$. Partition $\overrightarrow{W_M}$ into the following sets:

$$\overrightarrow{W_M}^{(+)} := \{\overrightarrow{w} \in \overrightarrow{W_M} \mid Cl(\overrightarrow{w}) < m \text{ and } \overrightarrow{s}(\overrightarrow{w}) = +1\} \quad \text{and}$$
$$\overrightarrow{W_M}^{(-)} := \{\overrightarrow{w} \in \overrightarrow{W_M} \mid Cl(\overrightarrow{w}) < m \text{ and } \overrightarrow{s}(\overrightarrow{w}) = -1\} \quad \text{and}$$
$$\overrightarrow{W_M}^{(m)} := \{\overrightarrow{w} \in \overrightarrow{W_M} \mid Cl(\overrightarrow{w}) = m\}.$$

The idea of this proof is that, in the sum of Proposition 6, the terms from $\overrightarrow{W_M}^{(+)} \cup \overrightarrow{W_M}^{(-)}$ cancel each other "almost" out, and the terms from $\overrightarrow{W_M}^{(m)}$ are small. To simplify the notation in the rest of the proof, let $y(\overrightarrow{x}, \overrightarrow{w}) := (y_F(\overrightarrow{x}, \overrightarrow{w}), y_G(\overrightarrow{x}, \overrightarrow{w}))$ and $H(y(\overrightarrow{x}, \overrightarrow{w})) := \widetilde{F}(y_F(\overrightarrow{x}, \overrightarrow{w})) \cdot G(y_G(\overrightarrow{x}, \overrightarrow{w}))$. Since $\widetilde{F}$ and G are continuous and bounded, so is H.

- Let $(\overrightarrow{x}, \overrightarrow{w})$ with $\overrightarrow{x} \subseteq \overrightarrow{w} \in \overrightarrow{W_M}^{(+)} \cup \overrightarrow{W_M}^{(-)}$. To any such $(\overrightarrow{x}, \overrightarrow{w})$ we associate a companion $(\overrightarrow{x}', \overrightarrow{w}')$ such that $\overrightarrow{w}'$ is the companion pairing of $\overrightarrow{w}$, and $\overrightarrow{x}' \subseteq \overrightarrow{w}'$ is obtained from $\overrightarrow{x}$ in the following way: if $\overrightarrow{w}'$ is obtained from $\overrightarrow{w}$ by replacing one or two triples (u, v, d), then $\overrightarrow{x}'$ is obtained by performing the same replacements on $\overrightarrow{x}$ (of course, only if $(u, v, d) \in \overrightarrow{x}$). Note that $y(\overrightarrow{x}, \overrightarrow{w})$ and $y(\overrightarrow{x}', \overrightarrow{w}')$ are equal in all components, except for one or two components, where they differ by at most $\frac{m}{N}$. Hence we have (for constant m) that

$$y(\overrightarrow{x}', \overrightarrow{w}') = y(\overrightarrow{x}, \overrightarrow{w}) + O(1/N) \cdot (1, \ldots, 1).$$

Since H is continuous, it follows

$$H(y(\overrightarrow{x}', \overrightarrow{w}')) = H(y(\overrightarrow{x}, \overrightarrow{w})) + O(1/N)\,.$$

– Let $(\overrightarrow{x}, \overrightarrow{w})$ with $\overrightarrow{x} \subseteq \overrightarrow{w} \in \overrightarrow{W_M}^{(m)}$. Note that all components of $y_F(\overrightarrow{x}, \overrightarrow{w})$ are at most $\frac{m}{N}$ or at least $1 - \frac{m}{N}$. Also note that for any vector $e \in \{0,1\}^{|\overrightarrow{x}|}$ it holds $H(e, y_G(\overrightarrow{x}, \overrightarrow{w})) = 0$. Since H is continuous, it follows

$$H(y(\overrightarrow{x}, \overrightarrow{w})) = O(1/N)\,.$$

Take $0 < \varepsilon < 1$. By Proposition 6 and the above considerations, we have:

$$\begin{aligned}
\mathbb{E}\mathbf{T} &= O(N^\varepsilon) + \frac{N^2}{r(1-r)} \sum_{\overrightarrow{w} \in \overrightarrow{W_M}} \overrightarrow{s}(\overrightarrow{w}) \sum_{\emptyset \neq \overrightarrow{x} \subseteq \overrightarrow{w}} H(y(\overrightarrow{x}, \overrightarrow{w})) \\
&= O(N^\varepsilon) + \frac{N^2}{r(1-r)} \cdot \left(\sum_{\overrightarrow{w} \in \overrightarrow{W_M}(+)} \sum_{\emptyset \neq \overrightarrow{x} \subseteq \overrightarrow{w}} H(y(\overrightarrow{x}, \overrightarrow{w})) \right. \\
&\qquad\qquad - \sum_{\emptyset \neq \overrightarrow{x}' \subseteq \overrightarrow{w}'} H(y(\overrightarrow{x}', \overrightarrow{w}')) \\
&\qquad\qquad \left. + \sum_{\overrightarrow{w} \in \overrightarrow{W_M}^{(m)}} \sum_{\emptyset \neq \overrightarrow{x} \subseteq \overrightarrow{w}} H(y(\overrightarrow{x}, \overrightarrow{w})) \right) \\
&= O(N^\varepsilon) + \frac{N^2}{r(1-r)} \cdot \left(\sum_{\overrightarrow{w} \in \overrightarrow{W_M}(+)} \sum_{\emptyset \neq \overrightarrow{x} \subseteq \overrightarrow{w}} O(1/N) \right. \\
&\qquad\qquad \left. + \sum_{\overrightarrow{w} \in \overrightarrow{W_M}^{(m)}} \sum_{\emptyset \neq \overrightarrow{x} \subseteq \overrightarrow{w}} O(1/N) \right) \\
&= O(N^\varepsilon) + O(N) = O(N)\,.
\end{aligned}$$

□

7 Conclusions and Future Work

We have obtained several results on the expected self-stabilization time $\mathbb{E}\mathbf{T}$ in Herman's algorithm. We have improved the best-known upper bound for arbitrary configurations, and we have given new and significantly better bounds for special classes of configurations: the full configuration, the random configuration, and, in particular, for configurations that arise from a fixed number of bit errors. For the latter class, $\mathbb{E}\mathbf{T}$ reduces to $O(N)$, pointing to a previously unknown feature that Herman's algorithm recovers quickly from bounded errors. We have also shown that an asynchronous version of Herman's algorithm not requiring synchronization behaves similarly. For our analysis, we have transferred techniques that were designed for the analysis of chemical reactions.

The conjecture of [16], saying that the equilateral configuration with three tokens constitutes the worst-case, remains open. We hope to exploit our closed-form expression for $\mathbb{E}\mathbf{T}$ to resolve this intriguing problem. While we have already shown that many relevant initial configurations provably converge faster, solving this conjecture would close the gap between the lower and upper bounds for stabilization time for arbitrary configurations. We would also like to investigate the performance of the algorithm in case the number of bit errors is not fixed, but is small (e.g., logarithmic) in the number of processes.

References

1. Balding, D.: Diffusion-reaction in one dimension. J. Appl. Prob. 25, 733–743 (1988)
2. PRISM case studies. Randomised self-stabilising algorithms, `http://www.prismmodelchecker.org/casestudies/self-stabilisation.php`
3. Cox, D., Miller, H.: The theory of stochastic processes. Chapman & Hall/CRC (2001)
4. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Commun. ACM 17(11), 643–644 (1974)
5. Dolev, S.: Self-Stabilization. MIT Press, Cambridge (2000)
6. Durrett, R., Kesten, H.: Random Walks, Brownian Motion and Interacting Particle Systems. Birkhauser Verlag AG, Basel (1991)
7. Feller, W.: An introduction to probability theory and its applications, vol. 1. John Wiley & Sons, Chichester (1968)
8. Flatebo, M., Datta, A.K.: Two-state self-stabilizing algorithms for token rings. IEEE Trans. Softw. Eng. 20(6), 500–504 (1994)
9. Fribourg, L., Messika, S., Picaronny, C.: Coupling and self-stabilization. Distributed Computing 18, 221–232 (2005)
10. Habib, S., Lindenberg, K., Lythe, G., Molina-Paris, C.: Diffusion-limited reaction in one dimension: Paired and unpaired nucleation. Journal of Chemical Physics 115, 73–89 (2001)
11. Herman, T.: Probabilistic self-stabilization. Information Processing Letters 35(2), 63–67 (1990), Technical Report at `ftp://ftp.math.uiowa.edu/pub/selfstab/H90.html`
12. Israeli, A., Jalfon, M.: Token management schemes and random walks yield self-stabilizing mutual exclusion. In: Proceedings of PODC 1990, pp. 119–131. ACM, New York (1990)
13. Kiefer, S., Murawski, A., Ouaknine, J., Worrell, J., Zhang, L.: On stabilization in Herman's algorithm. Technical report, arxiv.org (2011), `http://arxiv.org/abs/1104.3100`
14. Kutten, S., Patt-Shamir, B.: Stabilizing time-adaptive protocols. Theor. Comput. Sci. 220(1), 93–111 (1999)
15. Liggett, T.M.: Interacting particle systems. Springer, Heidelberg (2005)
16. McIver, A., Morgan, C.: An elementary proof that Herman's ring is $\theta(n^2)$. Inf. Process. Lett. 94(2), 79–84 (2005)
17. Nakata, T.: On the expected time for Herman's probabilistic self-stabilizing algorithm. Theoretical Computer Science 349(3), 475–483 (2005)
18. Schneider, M.: Self-stabilization. ACM Comput. Surv. 25(1), 45–67 (1993)

Online Graph Exploration: New Results on Old and New Algorithms*

Nicole Megow[1], Kurt Mehlhorn[1], and Pascal Schweitzer[2]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany.
{nmegow,mehlhorn}@mpi-inf.mpg.de
[2] The Australian National University, Canberra, Australia
pascal.schweitzer@anu.edu.au

Abstract. We study the problem of exploring an unknown undirected connected graph. Beginning in some start vertex, a searcher must visit each node of the graph by traversing edges. Upon visiting a vertex for the first time, the searcher learns all incident edges and their respective traversal costs. The goal is to find a tour of minimum total cost. Kalyanasundaram and Pruhs [23] proposed a sophisticated generalization of a Depth First Search that is 16-competitive on planar graphs. While the algorithm is feasible on arbitrary graphs, the question whether it has constant competitive ratio in general has remained open. Our main result is an involved lower bound construction that answers this question negatively. On the positive side, we prove that the algorithm has constant competitive ratio on any class of graphs with bounded genus. Furthermore, we provide a constant competitive algorithm for general graphs with a bounded number of distinct weights.

1 Introduction

In an exploration problem an agent, or searcher, has to construct a complete map of an environment without any a priori knowledge of its topology. The searcher makes all its decisions based on partial local knowledge and gathers new information on its exploration tour. Exploration problems appear in various contexts, such as robot motion planning in hazardous or inaccessible terrain, maintaining security of large networks, and searching, indexing, and analyzing digital data in the internet [7,20,28].

We study the online graph exploration problem on undirected connected graphs $G = (V, E)$. We assume that the vertices are labeled so that the searcher is able to distinguish them. Each edge $e = (u, v) \in E$ has a non-negative real weight $|e|$, also called the length or the cost of the edge. Beginning in a distinguished start vertex $s \in V$, the searcher learns G according to the following online paradigm, also known as *fixed graph scenario* [23]: whenever the searcher visits a vertex, it learns all incident edges, their weights, and the labels of their end vertices. To explore a new vertex, the searcher traverses previously learned edges in the graph. For traversing an edge, the searcher has to pay the respective edge cost. The task is to find a tour that visits all vertices V and returns to the start vertex. The goal is to find a tour of minimum total length. An illustration of this model (see [23]) is the scenario where vertices correspond to cities and

* Supported by the National Research Fund, Luxembourg, and cofunded under the Marie Curie Actions of the European Commission (FP7-COFUND).

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 478–489, 2011.

upon arrival in a city the searcher sees the road signs of routes to other cities including distance information.

A standard technique to measure the quality of online algorithms is *competitive analysis* [9], which compares the outcome of an algorithm with an *optimal offline solution*. For our graph exploration problem, the corresponding offline problem is the fundamental *Traveling Salesman Problem* (TSP), one of the most studied optimization problems which is in general even NP-hard to approximate [21]. It asks for a shortest tour that visits every vertex of a graph known in advance. For a positive number c, we call an online exploration algorithm *c-competitive*, if it computes for any instance a tour of total length at most c times the optimal offline tour through all vertices. The *competitive ratio* of an algorithm is the infimum over all c such that it is c-competitive.

The greedy algorithm *Nearest Neighbor* (NN) is a simple and fast heuristic that has been studied intensively in the traditional offline TSP environment. It repeatedly chooses the next vertex to be visited as an unexplored vertex closest to the current location. The worst case ratio for this greedy algorithm, $\Theta(\log n)$ [29], also applies to our online scenario. It is tight even on planar unit-weight graphs, which follows from a nice and simple lower bound construction of particular graphical instances [22].

In case all edges have equal weight, a *Depth First Search* (DFS) is 2-competitive. It yields a total tour not larger than twice the size of a minimum spanning tree (MST), a lower bound on the optimal tour. This is optimal in the unit-weight case [25].

For general graphs with arbitrary weights no constant competitive algorithm is known. A promising candidate was introduced by Kalyanasundaram and Pruhs [23]. Their algorithm ShortCut is a sophisticated generalization of DFS obtained by introducing a parameterized *blocking condition* that determines when to diverge from DFS. They prove an upper bound of 16 on its competitive ratio in planar graphs. The algorithm itself is defined for general graphs. However, since its introduction almost two decades ago, it has been open if ShortCut has constant competitive ratio in general. There has been no progress on this question since then, and in fact, all subsequent results concerned with our graph exploration setting only apply to simple cycles: In [2], it is shown that NN yields a competitive ratio of $3/2$ on simple cycles. Additionally, a lower bound of $5/4$ for any deterministic online algorithm is proven. Both lower and upper bound are improved in [25]. There, the authors give a more sophisticated algorithm which takes additionally the current total tour length into account. They prove that, on simple cycles, this algorithm achieves the best possible competitive ratio of $1 + \sqrt{3}$. It is not clear how the algorithm can be generalized and applied to more complex graphs.

Our contribution. We revisit algorithm ShortCut proposed by Kalyanasundaram and Pruhs [23]. We elaborate on the sophistication of the underlying idea, but report also a precarious issue in the given formal implementation. We propose a reformulation, which we call Blocking, highlighting the elaborate idea from [23], and adapt the proof of [23] to assure that the reformulation has constant competitive ratio for planar graphs. Here, a concise observation allows us to simplify the proof and to generalize it to graphs of bounded genus. More precisely, we generalize the upper bound on the competitive ratio of 16 for planar graphs to a bound of $16(1 + 2g)$ for graphs of genus at most g.

As further contribution we give a constant competitive algorithm for general graphs with a bounded number of distinct weights. Our online algorithm generalizes DFS to

an algorithm that hierarchically performs depth first searches on subgraphs induced by restricted weights. For arbitrary graphs with arbitrary weights we round weights up to the nearest power of 2 and apply the same algorithm. With this modification the algorithm has a competitive ratio of $\Theta(\log n)$ in general.

As our main result we show that Blocking does not have constant competitive ratio on general graphs. We use a classical construction of Erdős [15] of graphs with large girth and large minimum degree to construct complex graphs for which Blocking has arbitrarily large competitive ratio. Considering the fact that we have shown that Blocking is constant competitive for classes of graphs that have bounded genus, it seems plausible that similarly heavy machinery is indeed necessary for the lower bound construction.

Related work. Exploration problems have been studied extensively; see the surveys [7, 28]. In the sixties, such problems were addressed mainly from a game-theoretic perspective [19]. More recent research on online motion planning, aiming explicitly for worst-case performance guarantees on the total travel distance, was initiated in [5].

Generally, the geometry of the search environment can be arbitrary — a bounded or unbounded space, with or without obstacles, two, three, or higher dimensional. However, in many particular applications, it is possible to abstract from the geometry of the real environment and model the unknown search space as a graph, in which the searcher may only move along edges. First formal models for exploring an unknown graph were proposed in [27] in the context of finding a shortest path between two given points. Research on fully exploring a graph was initiated in [10]. In contrast to our problem, they consider the task of exploring all *edges* in a directed labeled unknown graph (with unit-weight edges). At any time, the searcher is given the number of unexplored edges leaving the vertex, but not their endpoints. Notice that the corresponding offline problem is the polynomially solvable Chinese postman problem, in contrast to the TSP in our setting. This exploration problem has been studied extensively in directed [1,16,24] and undirected graphs [11,26]. Numerous variants were considered, e.g., routing multiple searchers [6,14,17], models that impose additional constraints on the searcher, such as being tethered [12], or having a tank of limited capacity [4,8], and exploration problems in graphs without unique labeling but with some other additional information [18,20].

Our problem of exploring all *vertices* of a labeled undirected graph is in some sense also a variant of the initial problem in [10]. In particular, on trees the problem of exploring all vertices is equivalent to exploring all edges. Apart from the aforementioned previous work on our problem in planar graphs [23] and cycles [2,25], it has been studied on un-weighted trees also for multiple synchronously moving searchers [13,14,17].

Even though our online graph exploration problem has the classical TSP as corresponding offline problem, another class of online problems is typically regarded as *Online TSP* in the literature. In [3] a model is introduced in which the graph is given in advance and the vertices to be visited appear online over time. This means that new vertices appear as the salesman proceeds, in contrast to our model, independently of his current position. The corresponding offline problem is a TSP with release dates.

2 The Exploration Algorithm of Kalyanasundaram and Pruhs

We discuss the algorithm ShortCut, that was proposed and analyzed in [23].

Algorithm 1. The exploration algorithm $\mathsf{Blocking}_\delta(G, y)$

Input: A partially explored graph G, and a vertex y of G that is explored for the first time.

1: **while** there is an unblocked boundary edge $e = (u, v)$, with u explored and v unexplored, such that $u = y$ or such that e had previously been blocked by some edge (u', y) **do**
2: walk a shortest known path from y to u
3: traverse $e = (u, v)$
4: $\mathrm{Blocking}_\delta(G, v)$
5: walk a shortest known path from v to y
6: **end while**

Definition 1. *A vertex is* explored *once it has been visited by the searcher. An edge is* explored *once both endpoints are explored. A* boundary edge (u, v) *is an edge with an explored end vertex* u *and an unexplored end vertex* v.

We adopt the convention that for a boundary edge, the first entry is always vertex that has already been explored. For a set of edges E' we let $|E'| = \sum_{e \in E'} |e|$.

Algorithm ShortCut can be seen as a sophisticated variant of DFS. The crucial ingredient is a *blocking condition* depending on a fixed parameter $\delta > 0$, which determines when to diverge from DFS.

Definition 2. *At any point in time during the exploration of the graph, a boundary edge* $e = (u, v)$ *is said to be* blocked*, if there is a boundary edge* $e' = (u', v')$ *with* u' *explored and* v' *unexplored which is shorter than* e *(i.e,* $|e'| < |e|$*) and for which the length of any shortest known path from* u *to* v' *is at most* $(1 + \delta) \cdot |e|$.

Intuitively, the exploration algorithm ShortCut performs a standard DFS but traverses a boundary edge only if it is not blocked. Suppose the searcher is at a vertex u and considers traversing a boundary edge (u, v). If (u, v) is blocked then its traversal is postponed, possibly forever; otherwise the searcher traverses (u, v). Traversing (u, v) and exploring v may cause another edge (x, y), whose traversal was delayed earlier, to become unblocked. Then the shortest path from v to y is added as virtual edge (called jump edge in [23]) to the DFS-tree and can be traversed virtually like any real edge.

It is important to carefully update the blocking-state of edges as the algorithm proceeds. In particular, an edge which has become unblocked, after having previously been blocked, may become blocked again. This may be the case if a new shorter path from an unblocked edge to another boundary edge is revealed. In this case, the virtual edge must be removed again. Disregarding reblocking will cause an unbounded worst case ratio, even for planar graphs. This important issue is not explicitly addressed in the algorithm description in [23]. In particular, no means of removing virtual edges are provided therein.

In Algorithm 1, we formalize our interpretation of the algorithmic idea by Kalyanasundaram and Pruhs. To distinguish it from [23] and since the (parameterized) blocking condition is a very subtle and a key ingredient, we choose the name $\mathsf{Blocking}_\delta$.

To explore the entire graph starting in vertex s, we call Algorithm 1 as $\mathsf{Blocking}_\delta(G_s, s)$, where G_s is the partially explored graph in which only s has been visited so far.

We prove that Blocking$_\delta$ has constant competitive ratio on graphs of bounded genus. Our proof not only extends the one in [23] for planar graphs (genus 0), but also differs in using an additional argument which allows an easier handling of the recurrence.

Theorem 1. Blocking$_\delta$ *is* $2(2+\delta)(1+2/\delta)(1+2g)$*-competitive on graphs of genus* g*.*

Proof. Since in every iteration of the while loop a new vertex is explored, the algorithm terminates. It is easy to see that all vertices are eventually explored.

We let P be the set of edges that are traversed during some execution of Line 3.

For each iteration of the while loop, we charge all costs that occur in Lines 2, 3 and 5 to the edge in P traversed in Line 3. Since any execution of Line 3 explores a new vertex, every edge is charged in at most one while-loop iteration.

The cost charged to any edge e is at most $2(2+\delta)|e|$: Indeed, either the edge had previously not been blocked, in which case the cost is simply $2|e|$, or the edge e had previously been blocked by some edge ending in y, and therefore (by the definition of blocking) the distance from y to the starting point of e is at most $(1+\delta)\cdot|e|$. Thus Lines 2, 3 and 5 provoke costs of at most $(1+\delta)\cdot|e|$, $|e|$, and $(2+\delta)\cdot|e|$, respectively.

Let MST be a minimum spanning trees that shares a maximum number of edges with P. It suffices now to show that $|P| \le (1+2/\delta)(1+2g)|MST|$ in order to get an overall cost of at most $2(1+\delta)(1+2/\delta)(1+2g)|MST|$.

Claim. If an edge $e \in P \setminus MST$ is contained in a cycle C in $P \cup MST$, then the cycle C has length at least $(2+\delta)|e|$.

Proof. Suppose otherwise. On the cycle C, consider the edge $e' = (u,v) \in P \setminus MST$ with $|e'| \ge |e|$ that is charged the latest. W.l.o.g. suppose e' is traversed from u to v at the time it is charged. Due to the choice of MST, the edge e' is strictly larger than any edge in $C \cap MST$: Otherwise we could replace e' with an edge in MST to obtain a smaller minimum spanning tree or to obtain a minimum spanning tree that shares more edges with P. At the time e' is charged, e' is a boundary edge, and therefore not the whole cycle has been explored. Thus there is a boundary edge different from e' on the cycle. Moreover at this point in time e' is not blocked. Let e'' be the first boundary edge encountered when traversing $C - e'$ starting from u towards v. Since we assume the cycle has length less than $(2+\delta)|e| \le (2+\delta)|e'|$ and e' is not blocked, we conclude that e'' is not smaller than e' and thus not in MST. This contradicts the fact that e' is the edge in $P \setminus MST$ with $|e'| \ge |e|$ that is charged the latest and shows the claim. □

Consider an embedding of G on a surface of genus g. We choose $MST' \subseteq MST \cup P$ to be a maximal superset of MST obtained by repeatedly adding edges that do not separate two faces, i.e., are incident with only one face. (Topologically this can be viewed as adding a set of non-separating cycles, after contracting MST to a single point.) Since adding a non-separating edge increases the Euler characteristic of the surface bounded by the edges, and a surface of genus g has Euler characteristic $2-2g$, there are at most $2g$ edges in $MST' \setminus MST$. In case MST' does not bound a topological disk, we artificially add non-separating edges each of weight $|MST|$ to the graph induced by P, to obtain a superset MST'' of MST' that bounds a topological disk. These edges are artificial in the sense that they do not need to be edges of G. By the Euler characteristic argument above, there are at most $2g$ edges in $MST'' \setminus MST$ in total.

All edges in P, and thus, all edges in $MST'' \setminus MST$ have a weight not larger than $|MST|$, since otherwise they would be blocked until the whole minimum spanning tree has been explored. This implies $|MST''| \leq |MST| + 2g|MST|$. Since every edge $e \in P \setminus MST''$ is contained in a cycle of length at most $|e| + |MST|$ and edges in $MST'' \setminus MST'$ are of length $|MST|$, we can extend the claim: If an edge $e \in P \setminus MST''$ is contained in a cycle C in $P \cup MST''$, then the cycle C has length at least $(2+\delta)|e|$. We iteratively define for every edge $e \in P \setminus MST''$ a cycle C_e in the following way: In each step we choose an edge that together with edges in MST'' and edges to which a cycle has already been assigned closes a face cycle. Note that for two distinct edges $e, e' \in P \setminus MST''$ the associated cycles C_e and $C_{e'}$ are different. Every edge in $MST'' \cup P$ is contained in at most two such cycles, since they form a set of distinct face cycles. For an edge in $P \setminus MST''$ one of these cycles is C_e. In fact these cycles are exactly all face boundaries apart from the boundary of the outer face. Therefore, $|P \setminus MST''| \leq \frac{1}{1+\delta} \sum_{e \in P \setminus MST''} |C_e - e| \leq \frac{1}{1+\delta}(2|MST''| + |P \setminus MST''|)$, and thus $|P \setminus (MST'')| \leq (2/\delta)|MST''|$. We conclude that $|P| \leq (1 + 2/\delta)|MST''| \leq (1 + 2/\delta)(1 + 2g)|MST|$. Overall, we conclude that $\mathsf{Blocking}_\delta$ is $2(2+\delta)(1+2/\delta)(1+2g)$-competitive on graphs of genus g. □

Corollary 1. *Algorithm* $\mathsf{Blocking}_2$ *is* $16(1+2g)$*-competitive on graphs of genus* g*.*

3 A Lower Bound Construction

Our lower bound construction for Algorithm $\mathsf{Blocking}_\delta$ relies on a *base graph* H with specific bounds on girth and degree. Its existence is guaranteed by the following lemma which extends a classical construction of Erdős [15].

Lemma 1. *For all* $\bar{d}, \delta \in \mathbb{N}$ *there exists a connected bipartite graph* H *with minimum degree at least* $\bar{d}$*, maximum degree at most* $2\bar{d}$*, and a girth of* $g \geq \delta + 2$*.*

Let H be a connected bipartite n-vertex graph with average degree at least $\bar{d}$, maximum degree at most $2\bar{d}$, and girth at least $\delta + 2$ as given by Lemma 1. Suppose the partition classes have size n_1 and n_2. We fix orders $(u_1, \ldots, u_{n_1})$ and $(v_1, \ldots, v_{n_2})$ for the vertices in each of the bipartition classes. We call the vertices in the bipartition classes in-vertices and out-vertices, respectively. We order the edges by the lexicographical ordering satisfying $\{u, v\} < \{u', v'\}$ if $v < v'$ or ($v = v'$ and $u < u'$).

In H we now replace each in-vertex and each out-vertex by a release gadget and collection gadget, respectively. Our final construction will have edges with two types of weights, namely 1 and $w > 1$. We call edges of weight w *heavy* edges.

Description of the release gadgets: Figure 1 depicts a release gadget of degree 4. In general a release gadget consists of two parts, which we call left and right part. A gadget replacing a vertex of degree d consists in the left part of d vertices forming a path. Each of these vertices is attached to a heavy (red) edge of weight w that has an endpoint in some collection gadget. The right part contains d vertices forming a path. Each of these vertices is incident with an attached *release path* of length $d-1$. The endpoints of these

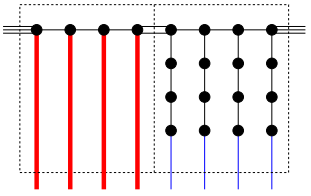


Fig. 1. Release gadget of a degree 4 vertex

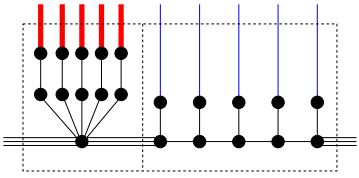


Fig. 2. Collection gadget of a degree 5 vertex

paths are incident with a (blue) edge that ends in a collection gadget. The two parts are joined by a *center path* of length $(\delta+1)w - d$ (depicted by a double edge) with unit-weight edges. Finally, each part has a *blocking path* (depicted as triple edges) of length $(\delta+1)w$ with unit-weight edges, by which it is connected to other release gadgets.

The crucial property of a release gadget is the following: The length of the center path is chosen such that the i-th heavy edge may be blocked by the first edge of the i-th release path, but not by the $(i+1)$-st release path (both times counting from the left to right). Once the exploration of a release path has begun, the algorithm will finish the exploration of the entire release path before exploring any other edges.

Thus, the i-th heavy edge of the gadget is blocked, if one of the release paths $1,\dots,i$ has not yet been explored. If a release path has been completely explored, we also say that the release has been triggered. Suppose in some release gadget all releases $1,\dots,i$ have been triggered, but the i-th heavy edge is still blocked. This situation implies that there is a path to some unit-weight boundary edge which exits the gadget via another heavy edge: Indeed, the blocking paths are sufficiently long to prevent other release gadgets from interfering with this fact. We will show later that at the moment release i is triggered such paths exiting via heavy edges do not exist.

It will be clear later that when a release gadget is entered for the first time, this happens via the blocking path to the right of the gadget. Assuming this for now, we can require that the online algorithm traverses the gadget from right to left, without entering the release paths: Indeed, whenever there is a choice among edges of equal weight, we can adversarially choose the edge that is traversed next.

Description of the collection gadgets: Figure 2 depicts a collection gadget of degree 5. In general a collection gadget consists of a left and a right part. For a gadget replacing a vertex of degree d, the left part has one vertex of degree d incident with d paths of length 2. The ends of these paths are incident with heavy (red) edges emanating from release gadgets. The right part of a collection gadget contains d vertices inducing a path. Each vertex on the path is adjacent to another vertex which itself is incident with a (blue) edge emanating from a right part of a release gadget. Three blocking paths (triple edges) of length $(\delta+1)w$ join the parts with each other and with other collection gadgets.

We will see that when a collection gadget is first entered, this happens via the blocking path to the left. We can then require adversarially that the online algorithm traverses from the left part directly to the right without exploring the left part. Then, on entering a vertex in the right, it deviates from the main path to explore the respective blue edge.

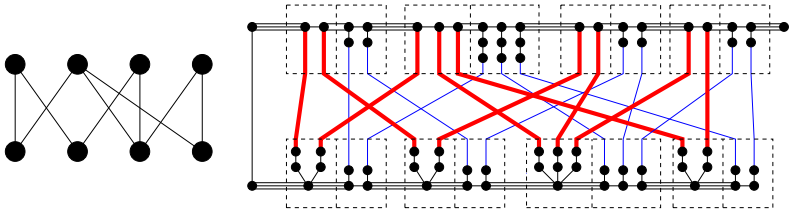


Fig. 3. The assembly of the gadgets: A base graph H (left), and the resulting graph with linked replacement gadgets (right) showing release gadgets on top and collection gadget at the bottom

We will argue that the algorithm will then return via a corresponding heavy edge. It then backtracks and subsequently explores the next vertex of the right part, and so on. Before leaving the gadget to the right, the gadget has been completely explored.

Assembly of the gadgets according to the base graph H: To assemble the gadgets using the base graph H (see Figure 3), we join the release gadgets according to the order of in-vertices along the blocking paths (triple lines). The same is done with the collection gadgets, with respect to the order of out-vertices. The right blocking path of the last (rightmost) release gadget is connected to a single vertex, the starting vertex, that we add to the graph. The left blocking path of the first (leftmost) release gadget and the first (leftmost) collection gadget are joined by two added, adjacent vertices.

The (red and blue) edges that run between the gadgets correspond to the edges in H. Heavy (red) edges of weight w run from a left part of a release gadget to a left part of a collection gadget. Blue edges of weight 1 run from a right part of a release gadget to a right part of a collection gadget.

In the lexicographical order of the edges defined above, we insert for each edge of H a heavy (red) edge and a blue edge. To insert a heavy edge corresponding to the edge (u, v) of H, we connect the leftmost unused vertex in the left part of the release gadget corresponding to u with the leftmost unused vertex in the left part of the collection gadget of v. To insert the blue edge, we connect the leftmost unused vertex in the right part of the release gadget corresponding to u with the leftmost unused vertex of the right part of the collection gadget of v.

Inserting the heavy edges in this ordering has the consequence that the ordering of the edges is exactly the ordering of their end vertices in the collection gadgets from left to right. Furthermore, within each release gadget, the heavy edges from left to right are also in the lexicographic order.

The tour traversed by the algorithm: Beginning at the starting vertex, we may require adversarially that the algorithm first traverses all release gadgets without exploring any release path. Then, via the two additional vertices on the left, the leftmost collection gadget is entered from the left, and the exploration continues into its right part. Subsequently release paths are triggered, one at a time. In the following we prove that the algorithm traverses all heavy edges of H. The lexicographic order defined on the edges is the order in which these edges are traversed. All of them are traversed from a release gadget to a collection gadget. The blue edges, each used to trigger a traversal of a heavy

edge, are traversed from a collection gadget to a release gadget. Recall that due to the length of the center path connecting the right and left part of a release gadget, a heavy edge is blocked, unless its corresponding release has been triggered.

Lemma 2. *The heavy (red) edges are traversed in the lexicographic order of the edges of the base graph. Whenever a release is triggered, the corresponding heavy edge e_r becomes unblocked and is explored subsequently.*

Proof. Inductively we assume that all release paths that correspond to edges that appear earlier than e_r in the ordering of edges have been completely explored, and all release paths that appear later than e_r are completely unexplored.

A heavy weight edge can only be blocked by an edge of weight 1. Thus, for a heavy edge to be blocked, there has to be a path of length at most $(\delta+1)w-1$ to a boundary edge of weight 1. To show the claim, we show that no such path exists for edge e_r.

To do so, we analyze where a hypothetical boundary edge of such a path may be situated in the graph. Observe that the length of blocking paths (triple edge) is chosen such that they cannot be traversed to reach a boundary edge within a distance of $(\delta+1)w-1$. Thus, only two possibilities have to be ruled out:

1. There is a path to a boundary edge that can be reached by a path of length $(1+\delta)w-1$, which traverses a center path (double edge).
2. There is a path to a boundary edge that uses heavy edges, but otherwise is completely contained in left parts of gadgets.

To rule out Possibility 1, observe that any path that uses a double line to cross from a left part of a release gadget to a right part, and then uses a complete release path is longer than $(\delta+1)w-1$. Moreover, since release paths are either completely explored or the corresponding heavy edge has by induction not been triggered, for every unexplored edge in the right part of a release gadget, all explored heavy edges in the left part are further away than $(\delta+1)w-1$.

To rule out possibility 2, note that the only boundary edges of weight 1 situated in the left part of a gadget are contained in the currently used collection gadget. All other left parts of gadgets have been completely explored or not explored at all. Thus, any path staying in the left parts of the gadgets that leads to a boundary edge in the left part of the currently used collection gadget will, together with e_r, project to a cycle in the graph H. Since the girth of H is at least $\delta+2$, the path has to use at least $\delta+1$ heavy edges and is thus of length more than $(\delta+1)\cdot w$.

We have shown that the heavy edge e_r becomes unblocked when its release is triggered. The algorithm thus explores e_r, returns to the release path of to e_r, backtracks, and continues to trigger the release corresponding to the next heavy edge. □

Theorem 2. *For no $\delta \in \mathbb{R}$ does* Blocking$_\delta$ *have constant competitive ratio.*

Proof. Consider a graph that is obtained from the replacement construction from a base graph H on n vertices with minimum degree $\bar{d}$, maximum degree at most $2\bar{d}$, and girth at least $\delta+2$ (Lemma 1). Including blocking paths, the number of unit-weight edges in a release gadget corresponding to a vertex v of degree $d(v)$ is $\mathcal{O}(d(v)^2)+\mathcal{O}(\delta w) \subseteq \mathcal{O}(\bar{d}^2)+\mathcal{O}(\delta w)$. This bound also holds for collection gadgets. Thus, for fixed δ, the

Algorithm 2. Exploration algorithm $\mathsf{hDFS}(G, u, w)$

Input: A partially explored graph G, a vertex u of G that is visited for the first time, and a weight $w \in \mathbb{R}_{\geq 0} \cup \{\infty\}$.

```
while there is a w' < w such that w' occurs in comp(G_{≤w'}, u) but comp(G_{≤w'}, u) is not completely explored do
    hDFS(G, u, w')
end while
choose a minimal spanning tree of comp(G_{<w}, u) and order all vertices according to a depth first search in this spanning tree
while there is a boundary edge (u', y') of weight w with u' ∈ comp(G_{<w}, u) do
    let (u', y') be a boundary edge of weight w with u' ∈ comp(G_{<w}, u) such that u' is minimal with respect to the ordering of comp(G_{<w}, u)
    traverse a shortest path to y'
    hDFS(G, y', w)
end while
traverse a shortest path to u
```

resulting graph has a minimum spanning tree of size $\mathcal{O}(n\bar{d}^2) + \mathcal{O}(nw)$. Since $\mathsf{Blocking}_\delta$ traverses all heavy edges (Lemma 2), it incurs a cost of $\Omega(\bar{d}nw)$. By choosing $\bar{d}$ large in comparison to all constants involved and then choosing w large in comparison to the constants and $\bar{d}^2$ the competitive ratio becomes arbitrarily large. □

4 Graphs with a Bounded Number of Distinct Weights

We describe a constant competitive algorithm for a bounded number of distinct weights.

Definition 3. *For any graph G, weight w, and vertex u, let $\mathrm{comp}(G_{\leq w}, u)$ be the connected component of the subgraph of G comprised of all edges of weight at most w containing u. The graph $\mathrm{comp}(G_{<w}, u)$ is defined similarly.*

Our algorithm *hierarchical depth first search* (hDFS), defined in Algorithm 2, explores $\mathrm{comp}(G_{\leq w}, u)$ for any weight $w \in \mathbb{R}_{\geq 0} \cup \{\infty\}$, which is provided as a parameter. The algorithm is based on a DFS in the graph $\mathrm{comp}(G_{\leq w}, u)$. However, whenever a new vertex of this component is encountered, it first explores $\mathrm{comp}(G_{<w}, u)$. The algorithm then intuitively simulates DFS in the graph $G/\mathrm{comp}(G_{<w}, u)$. Here G/H denotes the graph obtained from G by contracting the subgraph H of G to a single point. To ensure that the total length traversed within $H = \mathrm{comp}(G_{<w}, u)$ is not too large, the boundary edges leaving H are explored according to a specific order. This order is obtained by computing a depth first search on a minimum spanning tree of $\mathrm{comp}(G_{<w}, u)$.

The computation of $\mathrm{comp}(G_{<w}, u)$ can be reduced to recursive calls of the algorithm itself with parameters smaller than w due to the following basic observation:

Lemma 3. *The component $\mathrm{comp}(G_{<w}, u)$ is completely explored if and only if there is no boundary edge of weight smaller than w with an end-vertex in $\mathrm{comp}(G_{<w}, u)$.*

To explore the entire graph starting in vertex s, we call Algorithm 2 as $\mathsf{hDFS}(G_s, s, \infty)$, where G_s is the partially explored graph in which only s has been visited so far.

Theorem 3. hDFS *is $2k$-competitive on graphs with at most k distinct weights.*

Proof. We first prove that all vertices are explored. To prove this it suffices to show that hDFS(G, s, w) explores $\mathrm{comp}(G_{\leq w}, s)$. We show this by induction. Suppose there remains a boundary edge (u, v) with v unexplored after the call hDFS(G, s, w), and suppose this boundary edge is contained in $\mathrm{comp}(G_{\leq w}, s)$. By induction (u, v) has weight w. But u has been explored, thus there is a vertex y which was explored in a call with parameter w, such that this call caused u to be explored. But then the call hDFS(G, y, w) causes v to be explored, which gives a contradiction.

Let MST be a minimum spanning tree of G. To show $2k$-competitiveness, we show that for each $w < \infty$ the sum of all traversals made in calls with parameter w is at most $2|MST|$. For this it suffices to show: If F is a sub-forest of G that contains edges of weight at most w such that for each vertex u the graph $\mathrm{comp}(F_{<w}, u)$ is a minimum spanning tree of $\mathrm{comp}(G_{<w}, u)$, then F is contained in a minimum spanning tree of G. Finally note that the outer call with parameter $w = \infty$ does not incur any costs. □

For graphs with arbitrary weights, we adapt the algorithm by rounding each edge weight to the nearest power of 2 and simulating the exploration on this altered graph. This yields a competitive ratio of $\Theta(\log(n))$ for graphs with n vertices.

5 Concluding Remarks

Our main result is a non-trivial graph construction which proves that Algorithm Blocking does not have constant competitive ratio on arbitrary graphs. This answers a longstanding open question. Nevertheless, the result does not generally rule out online algorithms with constant competitive ratio. In particular, our construction involves only two distinct types of weights, and thus, our new Algorithm hDFS has constant competitive ratio. However, at present, there is no candidate for an algorithm that may achieve a constant competitive ratio on general graphs. Of course showing that no such algorithm exists might require a construction even more complicated than the one presented in this paper. For such result it might be helpful to use the fact that one can equivalently consider the exploration model in which the label of a vertex is only revealed upon arrival at the vertex. This can be seen by replacing each vertex by a star with edges of small weight, and linking the previous neighbors to the outer vertices of the star.

References

1. Albers, S., Henzinger, M.R.: Exploring unknown environments. SIAM J. Comput. 29(4), 1164–1188 (2000)
2. Asahiro, Y., Miyano, E., Miyazaki, S., Yoshimuta, T.: Weighted nearest neighbor algorithms for the graph exploration problem on cycles. Inf. Process. Lett. 110(3), 93–98 (2010)
3. Ausiello, G., Feuerstein, E., Leonardi, S., Stougie, L., Talamo, M.: Algorithms for the on-line travelling salesman. Algorithmica 29(4), 560–581 (2001)
4. Awerbuch, B., Betke, M., Rivest, R.L., Singh, M.: Piecemeal graph exploration by a mobile robot. Inf. Comput. 152(2), 155–172 (1999)

5. Baeza-Yates, R.A., Culberson, J.C., Rawlins, G.J.E.: Searching in the plane. Information and Computation 106(2), 234–252 (1993)
6. Bender, M.A., Slonim, D.K.: The power of team exploration: Two robots can learn unlabeled directed graphs. In: Proceedings of FOCS, pp. 75–85 (1994)
7. Berman, P.: On-line searching and navigation. In: Fiat, A. (ed.) Dagstuhl Seminar 1996. LNCS, vol. 1442, pp. 232–241. Springer, Heidelberg (1998)
8. Betke, M., Rivest, R.L., Singh, M.: Piecemeal learning of an unknown environment. Machine Learning 18, 231–254 (1995)
9. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
10. Deng, X., Papadimitriou, C.H.: Exploring an unknown graph (extended abstract). In: Proceedings of FOCS, pp. 355–361 (1990)
11. Dessmark, A., Pelc, A.: Optimal graph exploration without good maps. Theoretical Computer Science 326(1-3), 343–362 (2004)
12. Duncan, C.A., Kobourov, S.G., Kumar, V.S.A.: Optimal constrained graph exploration. ACM Trans. Algorithms 2, 380–402 (2006)
13. Dynia, M., Kutyłowski, J., der Heide, F.M.a., Schindelhauer, C.: Smart robot teams exploring sparse trees. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 327–338. Springer, Heidelberg (2006)
14. Dynia, M., Lopuszanski, J., Schindelhauer, C.: Why robots need maps. In: Prencipe, G., Zaks, S. (eds.) SIROCCO 2007. LNCS, vol. 4474, pp. 41–50. Springer, Heidelberg (2007)
15. Erdős, P.: Graph theory and probability. Canad. J. Math. 11, 34–38 (1959)
16. Fleischer, R., Trippen, G.: Exploring an unknown graph efficiently. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 11–22. Springer, Heidelberg (2005)
17. Fraigniaud, P., Gąsieniec, L., Kowalski, D.R., Pelc, A.: Collective tree exploration. Netw. 48, 166–177 (2006)
18. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Impact of memory size on graph exploration capability. Discrete Applied Mathematics 156(12), 2310–2319 (2008)
19. Gal, S.: Search Games. Academic Press, London (1980)
20. Gąsieniec, L., Radzik, T.: Memory efficient anonymous graph exploration. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG 2008. LNCS, vol. 5344, pp. 14–29. Springer, Heidelberg (2008)
21. Gutin, G., Punnen, A.P.: The Traveling Salesman Problem and Its Variations. Springer, Heidelberg (2002)
22. Hurkens, C.A.J., Woeginger, G.J.: On the nearest neighbor rule for the traveling salesman problem. Operations Research Letters 32(1), 1–4 (2004)
23. Kalyanasundaram, B., Pruhs, K.: Constructing competitive tours from local information. Theor. Comput. Sci. 130(1), 125–138 (1994)
24. Kwek, S.: On a simple depth-first search strategy for exploring unknown graphs. In: Rau-Chaplin, A., Dehne, F., Sack, J.-R., Tamassia, R. (eds.) WADS 1997. LNCS, vol. 1272, pp. 345–353. Springer, Heidelberg (1997)
25. Miyazaki, S., Morimoto, N., Okabe, Y.: The online graph exploration problem on restricted graphs. IEICE Transactions on Information and Systems E92.D(9), 1620–1627 (2009)
26. Panaite, P., Pelc, A.: Exploring unknown undirected graphs. Journal of Algorithms 33(2), 281–295 (1999)
27. Papadimitriou, C., Yannakakis, M.: Shortest paths without a map. Theoretical Computer Science 84(1), 127–150 (1991)
28. Rao, N., Kareti, S., Shi, W., Iyengar, S.: Robot navigation in unknown terrains: Introductory survey of nonheuristic algorithms. Report ORNL/TM-12410, Oak Ridge Nat. Lab (1993)
29. Rosenkrantz, D.J., Stearns, R.E., Lewis II, P.M.: An analysis of several heuristics for the traveling salesman problem. SIAM J. Comput. 6(3), 563–581 (1977)

Distance Oracles for Vertex-Labeled Graphs

Danny Hermelin[1], Avivit Levy[2], Oren Weimann[3], and Raphael Yuster[4]

[1] Max-Planck Institut fur informatik
hermelin@mpi-inf.mpg.de
[2] Shenkar College and CRI at University of Haifa
avivitlevy@shenkar.ac.il
[3] Weizmann Institute
oren.weimann@weizmann.ac.il
[4] University of Haifa
raphy@math.haifa.ac.il

Abstract. Given a graph $G = (V, E)$ with non-negative edge lengths whose vertices are assigned a *label* from $L = \{\lambda_1, \ldots, \lambda_\ell\}$, we construct a compact *distance oracle* that answers queries of the form: "What is $\delta(v, \lambda)$?", where $v \in V$ is a vertex in the graph, $\lambda \in L$ a vertex label, and $\delta(v, \lambda)$ is the distance (length of a shortest path) between v and the closest vertex labeled λ in G. We formalize this natural problem and provide a hierarchy of *approximate distance oracles* that require subquadratic space and return a distance of constant stretch. We also extend our solution to dynamic oracles that handle label changes in sublinear time.

1 Introduction

In this paper we consider an *all-pairs shortest paths* variant on vertex-labeled graphs: We are given an undirected graph $G = (V, E)$ with $m = |E|$ edges and $n = |V|$ vertices. Each edge is assigned a nonnegative length and each vertex is assigned a *label*, given as a function $\lambda : V \to L$, where $L = \{\lambda_1, \ldots, \lambda_\ell\}$ is a set of $\ell \leq n$ distinct labels. The goal is to preprocess G in order to answer *vertex-label distance queries*, *i.e.* queries of the form: "What is $\delta(v, \lambda)$?", where $v \in V$ is a vertex in the graph, $\lambda \in L$ a label, and $\delta(v, \lambda)$ is the distance (length of a shortest path) between v and the closest vertex labeled λ in G.

Vertex-labeled graphs are commonly used to analyze behaviors and structures of networks. Typically, labels specify a common functionality of a set of nodes in the network. Often in such networks we are not interested in shortest paths between two specific vertices, but rather from a specific vertex to any vertex that can provide a specific function. For instance, in a computer network labels might indicate different types of servers in the network, while in an automotive network they can indicate different services provided for drivers on the road. Common type of queries in these types of networks include queries such as "What is the closest FTP server?" or "Where is the closest McDonald's?"

A simple solution for the above problem is to construct a table indexed by vertex-label pairs, storing at entry (v, λ) the distance $\delta(v, \lambda)$, thus allowing

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 490–501, 2011.

queries to be answered in $O(1)$ time. Such a table can be constructed with ℓ calls to a standard single-source shortest-paths algorithm such as Dijkstra's Algorithm. This is done by constructing for each label $\lambda \in L$, an auxiliary graph G_λ, created by removing all vertices labeled λ from G and adding a new vertex v_λ which is adjacent to all neighbors of the removed vertices (using appropriate minimum edge-lengths). Running Dijkstra's Algorithm in G_λ from v_λ gives all distances in G to vertices labeled λ. The total running time of this construction is $O(m\ell)$ and the space required for the table is $O(n\ell)$.

There are two problems with this solution. First, the $O(n\ell)$ space requirement is unacceptable in many cases, especially when the number of labels ℓ in the graph is quite large. It is therefore beneficial in these cases to replace this solution by a more compact data structure, perhaps at the cost of providing only approximate distances. Such data structures have been coined *approximate distance oracles* in the literature. The second problem with the data structure is its inability so support efficient changes of labels. A change of a label can reflect a change in its functionality or in availability of services in the network. Indeed, even a single label change may incur $\Omega(n)$ changes in the data structure.

Distance oracles for unlabeled graphs have been studied quite extensively. A seminal result of Thorup and Zwick [9] achieves for any given integer $k \geq 2$, an $O(kn^{1+1/k})$ space oracle which returns distances with at most $(2k-1)$ *stretch* in $O(k)$ time. That is, given an input pair of vertices (u, v), the oracle outputs a distance $\mathbf{dist}(u, v)$ with $\delta(u, v) \leq \mathbf{dist}(u, v) \leq (2k-1) \cdot \delta(u, v)$, where $\delta(u, v)$ is the actual distance between u and v. An earlier observation by Matoušek [4] shows that this is essentially optimal for any k, assuming a conjecture of Erdős [3] concerning the girth of undirected graphs. A derandomization of the oracle was presented in [6], and faster construction times where presented in [2] and [1] with a slight increase of the promised stretch. Recently, Pătraşcu and Roditty [5] showed how to construct, for unweighted graphs, an oracle of size $O(n^{5/3})$ that, when queried about a pair of vertices (u, v), returns in $O(1)$-time a distance $\mathbf{dist}(u, v)$ bounded by $\delta(u, v) \leq \mathbf{dist}(u, v) \leq 2\delta(u, v) + 1$. For further results see also [5,9].

Applying the existing approximate distance oracles to labeled graphs faces several obstacles. First of all, one might be tempted to construct an oracle for the complete bipartite graph $B(G)$ formed on the vertex sets V and L, where the length of the edge $\{v, \lambda\}$ is set to $\delta(v, \lambda)$. However, the oracle constructed for this graph might output false distances, as paths in $B(G)$ using an intermediate vertex $\lambda \in L$ do not occur in G. Similarly, the solution that adds a new vertex for each label, connecting it with zero weight edges to all vertices with the same label, fails. Another option is to build ℓ different oracles, one per each label. This unfortunately yields a solution that requires more space than the naive $O(n\ell)$ solution described above, even when using the optimal construction of Thorup and Zwick. Thus, "black-box" solutions such as these are destined for failure.

The next obvious approach is to hack and modify the existing oracles, adapting them to support vertex-label queries. However, this has difficulties as well. All known oracles achieving close to optimal stretch are highly dependent on

knowing both the source and the target vertex. For instance, Thorup and Zwick's $(2k-1)$-stretch oracles find the path between the pair of query vertices by advancing from both of them simultaneously. This obviously cannot be applied to vertex-label queries, as we only know the identity of the source vertex in our query. There is, however, another oracle scheme by Thorup and Zwick [8] designed for routing that does not switch between query vertices. Oracles produced by this scheme require $O(kn^{1+1/k})$ space, and answer queries with $4k-5$ stretch in $O(k)$ time. We show in Section 2 how to adapt this scheme to vertex-label oracles achieving the same performance.

Theorem 1 (Thorup-Zwick Vertex-Label Distance Oracles). *A vertex-label distance oracle of expected size $O(kn^{1+1/k})$ with stretch $(4k-5)$ and query time $O(k)$ can be constructed in $O(kmn^{1/k})$ time.*

There are two problems with the oracles given by Theorem 1. First, the expected space of the oracles depends only on the number of vertices in the graph, and not on the number of labels. This might be too much when ℓ is significantly smaller than n. For instance, when say $\ell = O(\sqrt{n})$, the trivial $O(n\ell)$ solution discussed above gives an $O(n^{3/2})$-space oracle which produces exact distances, as opposed to the stretch-3 distances returned by the oracle of Theorem 1. This problem becomes even more apparent when $\ell = \text{polylog}(n)$. The second problem is that the oracles given in Theorem 1 cannot be adapted to support dynamic labels efficiently. Indeed, even a single label change requires in the worst case reconstruction of almost the entire oracle from scratch.

In Section 3, we address the first issue raised above, and give a first step towards the second issue as well. We show how to construct an alternative scheme of vertex-label distance oracles which have expected space bounds depending on both ℓ and n, thereby achieving better bounds than the scheme of Theorem 1. While the stretch of this scheme grows exponentially in k, we are able to obtain the *same* stretch as in Theorem 1 for the cases of $k = 2$ and $k = 3$. Furthermore, for the case of $\ell = \text{polylog}(n)$, our scheme gives an $O(n \lg n)$-space oracle with constant stretch, while Theorem 1 cannot achieve even poly-logarithmic stretch within comparable space bounds.

Theorem 2 (Compact Vertex-Label Distance Oracles). *A vertex-label distance oracle of expected size $O(kn\ell^{1/k})$ with stretch (2^k-1) and query time $O(k)$ can be constructed in $O(kmn^{k/(2k-1)})$ time.*

The oracle schemes given by both theorems above still do not support label changes. Nevertheless, in Section 4 we show how the scheme of Theorem 2 can be modified in a way that allows oracles supporting such updates in sublinear time. This results in oracles with size asymptotically the same as the ones given by Theorem 1, and with stretch slightly bigger then the stretch of the oracles given in Theorem 2.

Theorem 3 (Dynamic Vertex-Label Distance Oracles). *A vertex-label distance oracle of expected size $O(kn^{1+1/k})$ with stretch $(2 \cdot 3^{k-1}+1)$ can support label changes in $O(kn^{1/k} \lg n)$ time and queries in $O(k)$ time.*

We also remark that it is possible to combine our construction with the ideas of Pătraşcu and Roditty [5] to obtain a stretch-3 oracle of expected size $O(n^{5/3})$, supporting label changes in in $O(n^{2/3} \lg n)$ time (see Section 4).

2 Adaptation of Thorup-Zwick Oracles

In this section we outline a simple adaptation of Thorup and Zwick's [9] result that supports vertex-label distance queries, thus providing a proof for Theorem 1. Our adaptation is based on an alternative query algorithm given by Thorup and Zwick in [8]. We will use the following notation throughout the section, and the remainder of the paper. For a pair of vertices u and v, we let $\delta(u, v)$ denote the distance between u to v, where $\delta(u, v) = \infty$ if there is no path connecting them. For a non-empty vertex-subset $S \subseteq V$, we let $\delta(v, S) := \min_{u \in S} \delta(u, v)$, and we set $\delta(v, \emptyset) := \infty$. For a label $\lambda \in L$, we denote by $\delta(v, \lambda)$ the distance $\delta(v, V_\lambda)$, where $V_\lambda := \{v \in V : \lambda(v) = \lambda\}$.

The data structure. For a given positive integer k, the preprocessing algorithm of Thorup and Zwick constructs the sets $V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_{k-1} \supseteq A_k = \emptyset$. These sets are referred to as the levels of G, and a vertex v is said to be in *level* i if $v \in A_i$. The i-th level A_i is constructed by sampling vertices of A_{i-1} independently at random, taking a vertex $v \in A_{i-1}$ to A_i with probability $n^{-1/k}$. For $i \in \{1, \ldots, k-1\}$, the *i-th pivot* of a vertex $v \in V$, denoted $p_i(v)$, is defined to be the vertex closest to v in the i-th level of G. That is, $\delta(v, p_i(v)) = \delta(v, A_i)$. We also set $p_0(v) := v$. The oracle stores for each vertex v, the identity of each of its k pivots, along with the k distances $\delta(v, p_i(v))$. In addition, v stores the distances to all vertices in its *bunch* $B(v)$, where

$$B(v) := \bigcup_{i=0}^{k-1} \big\{u \in A_i \setminus A_{i+1} : \delta(v, u) < \delta(v, A_{i+1})\big\}.$$

We next describe the additional information that our adaptation requires. For a label $\lambda \in L$, define the bunch $B(\lambda)$ by $B(\lambda) := \bigcup_{v \in V_\lambda} B(v)$. Now for every vertex $u \in B(\lambda)$, we define u_λ to be the λ-labeled vertex closest to u and that satisfies $u \in B(u_\lambda)$. For each $\lambda \in L$ and $u \in B(\lambda)$, we store the distance $\delta(u, u_\lambda)$. Note that each one of these distances is computed by the original Thorup-Zwick construction. Furthermore, the additional space required by our construction does not change the asymptotic size of the original oracles, since

$$\sum_{\lambda \in L} |B(\lambda)| = \sum_{\lambda \in L} \Big| \bigcup_{v \in V_\lambda} B(v) \Big| \leq \sum_{\lambda \in L} \sum_{v \in V_\lambda} |B(v)| = \sum_{v \in V} |B(v)|.$$

Adapted query algorithm. Our adapted query algorithm examines each of the k vertices $p_i(v)$ for $0 \leq i < k$, starting with $p_0(v) := v$. For each such vertex $w := p_i(v)$, we check if $w \in B(\lambda)$. If so, we declare i to be a *valid* index and we compute the distance $\delta(v, w) + \delta(w, w_\lambda)$ in $O(1)$ time (via hash tables). After $\Theta(k)$ time, our query algorithm returns the distance

$$\mathbf{dist}(v, \lambda) := \min\{\delta(v, w) + \delta(w, w_\lambda) : w = p_i(v) \text{ and } i \text{ is valid }\}.$$

The important difference between our query algorithm and that of [8] is that we need to check *all* valid indices while [8] can stop when it reaches the first valid index. In other words, the query algorithm of [8] upon query (v, u) returns $\mathbf{dist}(v, u) = \delta(v, w) + \delta(w, u)$ where $w = p_i(v)$ for the smallest i such that $w = p_i(v) \in B(u)$. They show (Lemma A.1 in [8]) that the stretch is then bounded by $\mathbf{dist}(v, u) \leq (4k - 3) \cdot \delta(v, u)$. In our settings, let u be the (unknown) λ-labeled vertex such that $\delta(v, u) = \delta(v, \lambda)$. When we discover the first $w = p_i(v) \in B(\lambda)$, we cannot promise that $\delta(w, w_\lambda) \leq \delta(w, u)$ since maybe $w \notin B(u)$. We can however guarantee, from the definition of $B(\lambda)$, that for *some* i we will have $w = p_i(v) \in B(u)$. We therefore have that $\mathbf{dist}(v, \lambda) \leq (4k - 3) \cdot \delta(v, \lambda)$. Finally, the same argument used in [8] (Lemma A.2) shows how the stretch bound can be reduced from $(4k - 3)$ to $(4k - 5)$, thus proving Theorem 1.

3 More Compact Oracles

In what follows we describe our vertex-label distance oracles for vertex-labeled graphs. In particular, we provide a complete proof of Theorem 2.

3.1 The Data Structure

The first step in constructing our $(2^k - 1)$-stretch oracle is similar to the Thorup-Zwick adaptation of Section 2. For a given positive integer k, we first construct the sets $V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_{k-1} \supseteq A_k = \emptyset$ which will form the levels of G. However, unlike the construction of Section 2, we select the vertices into levels with probability depending on ℓ. That is, the i-th level A_i is constructed by sampling vertices of A_{i-1} independently at random, taking a vertex $v \in A_{i-1}$ to A_i with probability $\ell^{-1/k}$. Thus, for any $v \in V$, the probability that $v \in A_i$ for some $i \in \{0, \ldots, k - 1\}$, is exactly $\ell^{-i/k}$. The following bound on the expected size of A_i follows immediately:

Lemma 1. $\mathbb{E}[|A_i|] = n\ell^{-i/k}$ *for each* $i \in \{0, \ldots, k - 1\}$.

The idea of sampling vertices with probability independent of n was already suggested by Roditty, Thorup, and Zwick in [6]. The problem they considered was a distance oracle that answers $\delta(u, v)$ queries where u can be any vertex but v is known to belong to some subset $S \subset V$. For this problem, they showed that if we sample vertices with probability $|S|^{-1/k}$ then the original Thorup-Zwick oracle works and requires only $O(n|S|^{-1/k})$ space. For our problem however, the original Thorup-Zwick oracle can not be made to work if we sample with probability $\ell^{-1/k}$. This is because we are not dealing with a set of vertices but rather with a set of sets (the set of labels where each label is a set of vertices). We now show how to overcome this by presenting a different oracle that in particular uses *balls* instead of *bunches*.

For a vertex $v \in A_i \setminus A_{i+1}$, we define the *ball of* v to be the set of labels $B(v)$ of *all* vertices in G that are closer to v than A_{i+1}. That is, $B(v) = \{\lambda(u) : \delta(u, v) <$

$\delta(v, A_{i+1})\}$. Notice the difference between our *balls* and Thorup-Zwick's *bunches*. Each vertex stores all the labels in its ball in an appropriate hash table which allows us to determine in $O(1)$ time whether $\lambda \in B(v)$, for any label $\lambda \in L$. Furthermore, we store the distances $\delta(v, \lambda)$ to each label $\lambda \in B(v)$, allowing $O(1)$ answers to vertex-label queries for labels appearing inside the ball of the query vertex. Note that as $A_k = \emptyset$, we have $\delta(v, A_k) = \infty$ for all $v \in V$, and so $B(v) = L$ for any vertex $v \in A_{k-1}$. Nevertheless, the following lemma shows that the expected number of labels is not big at vertices appearing in lower levels.

Lemma 2. $\mathbb{E}[|B(v)|] \leq \ell^{(i+1)/k}$ *for any vertex* $v \in A_i \setminus A_{i+1}$, $i \in \{0, \ldots, k-1\}$.

To complete the description of our distance oracle, we assign routers to the vertices in our graph. For a vertex $v \in A_i \setminus A_{i+1}$, $i \in \{0, \ldots, k-2\}$, we let the *router of* v, denoted $r(v)$, be a vertex for which $\delta(v, r(v)) = \delta(v, A_{i+1})$. That is, $r(v)$ is the closest vertex to v at the next level of G. Since vertices at level $k-1$ have no vertices at the next level, we set $r(v) = v$ for all $v \in A_{k-1}$. Along with the ball of labels $B(v)$ stored at each vertex $v \in V$, our distance oracle also stores at v the identity of its router $r(v)$, together with the distance $\delta(v, r(v))$. Thus, the total space required by our data structure is (asymptotically) the total sizes of the balls $B(v)$, which can easily be bounded using Lemma 1 and Lemma 2.

Lemma 3. *The expected space of our data structure is* $O(kn\ell^{1/k})$.

3.2 Vertex-Label Queries

We next proceed to describe how a vertex-label query is processed. Let $(v \in V, \lambda \in L)$ denote an input vertex-label pair. The query algorithm starts by determining whether λ is in the ball of v. If so, the exact distance $\delta(v, \lambda)$ is retrieved immediately. Otherwise, it hops to the router of v, and continues the search there. If $\lambda \notin B(r(v))$, the algorithm hops to the router of $r(v)$, and so forth.

To be more precise, let us introduce the following notation: For $i \in \{0, \ldots, k-1\}$, we let r_i denote the vertex $r^{(i)}(v)$, where $r^{(i)}$ is the function resulting in concatenating r with itself i times. That is, $r_i = r(r_{i-1})$ for $i \in \{1, \ldots, k-1\}$, and $r_0 = v$. (Again, notice the difference between these routers and the pivots of Section 2.) The query algorithm determines the smallest integer $i_0 \in \{0, \ldots, k-1\}$ for which $\lambda \in B(r_{i_0})$, and returns the distance

$$\mathbf{dist}(v, \lambda) := \sum_{0 \leq i < i_0} \delta(r_i, r_{i+1}) + \delta(r_{i_0}, \lambda).$$

See Figure 1 for an example. Note that as $r_{k-1} \in A_{k-1}$, we have $\lambda \in B(r_{k-1})$, and so i_0 is well-defined. Furthermore, the path from v to r_{i_0}, and then from r_{i_0} to a vertex labeled λ, gives a path from v to a vertex labeled λ as required.

Determining the smallest integer i_0 for which $\lambda \in B(r_{i_0})$ takes $O(k)$ time, by iteratively hopping through the r_i's. Furthermore, as the distances $\delta(r_i, r_{i+1})$, for $i \in \{0, \ldots, i_0 - 1\}$, have been stored by our data-structure, along with the

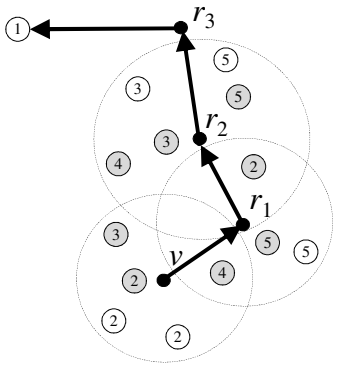


Fig. 1. An example of the query procedure for $k = 4$. The input to the query is $(v, 1)$, and the arrows depict the output path. The dashed circles around v, r_1, r_2, and r_3 represent the balls around them. The gray colored vertices are the vertices which are stored at each ball (the distances are assumed to be Euclidean).

distance $\delta(r_{i_0}, \lambda)$, we can report the resulting distance $\mathbf{dist}(v, \lambda)$ in $O(k)$ time as well. This gives us the promised $O(k)$ query time of Theorem 2. The next lemma shows that the stretch of our query is also as stated in Theorem 2:

Lemma 4. $\delta(v, \lambda) \leq \mathbf{dist}(v, \lambda) \leq (2^k - 1) \cdot \delta(v, \lambda)$ *for all* $(v, \lambda) \in V \times L$.

Proof. Let $(v, \lambda) \in V \times L$, and let $i_0 \in \{0, \ldots, k-1\}$ denote the smallest integer for which $\lambda \in B(r_{i_0})$. Then $\mathbf{dist}(v, \lambda) := \sum_{0 \leq i < i_0} \delta(r_i, r_{i+1}) + \delta(r_{i_0}, \lambda)$, where $r_0 := v$ and $r_i := r(r_{i-1})$ for all $i \in \{1, \ldots, k-1\}$. The lower bound $\delta(v, \lambda) \leq \mathbf{dist}(v, \lambda)$ in the lemma follows from the fact that $\mathbf{dist}(v, \lambda)$ is the length of an actual path from v to a vertex labeled λ. The proof of the upper bound relies on the following crucial inequality which follows from our definition of the balls $B(v)$, and from the fact that $\lambda \notin B(r_i)$ for all $i \in \{0, \ldots, i_0 - 1\}$:

$$\delta(r_i, r_{i+1}) \;\leq\; \delta(r_i, \lambda) \text{ for all } i \in \{0, \ldots, i_0 - 1\}. \tag{1}$$

As an intermediate step in proving the upper bound, we use (1) to prove inequality (2) below by induction on i:

$$\delta(r_i, \lambda) \leq\; 2^i \cdot \delta(v, \lambda) \text{ for all } i \in \{0, \ldots, i_0\}. \tag{2}$$

For $i = 0$, we have $\delta(r_0, \lambda) = \delta(v, \lambda)$ so (2) holds. Assume therefore that $i > 0$, and that (2) holds for all $j < i$. By the triangle-inequality, we get the following bound $\delta(r_i, \lambda) \leq \sum_{0 \leq j < i} \delta(r_j, r_{j+1}) + \delta(v, \lambda)$. Thus, by (1) and our inductive hypothesis we have:

$$\begin{aligned}
\delta(r_i, \lambda) &\leq \textstyle\sum_{0 \leq j < i} \delta(r_j, r_{j+1}) + \delta(v, \lambda) \\
&\leq \textstyle\sum_{0 \leq j < i} \delta(r_j, \lambda) + \delta(v, \lambda) \\
&\leq \textstyle\sum_{0 \leq j < i} 2^j \cdot \delta(v, \lambda) + \delta(v, \lambda) \\
&= 2^i \cdot \delta(v, \lambda).
\end{aligned}$$

Inequality (2) therefore holds. Now, using (1) and (2) the upper bound easily follows as

$$\begin{aligned}\mathbf{dist}(v,\lambda) &= \textstyle\sum_{0\le i<i_0} \delta(r_i, r_{i+1}) + \delta(r_{i_0}, \lambda) \\ &\le \textstyle\sum_{0\le i<i_0} \delta(r_i, \lambda) + \delta(r_{i_0}, \lambda) \\ &\le \textstyle\sum_{0\le i<i_0} 2^i \cdot \delta(v,\lambda) + 2^{i_0} \cdot \delta(v,\lambda) \\ &= (2^{i_0+1} - 1) \cdot \delta(v,\lambda).\end{aligned}$$

This proves the upper bound in the lemma, since $i_0 \le k-1$. □

3.3 Construction

The time to construct the data structure is composed of two parts. The first is the time to find for every vertex v the distance from v to all vertices $\widehat{B}(v) = \{u : \delta(u,v) < \delta(v, A_{i+1})\}$. Observe that $B(v)$ can be immediately obtained from $\widehat{B}(v)$. A simple modification of Dijkstra's Algorithm starting from source v computes $\widehat{B}(v)$ by inspecting only $|\widehat{B}(v)|$ vertices. The inspected edges are only edges (u,w) where $u \in \widehat{B}(v)$ *or* $w \in \widehat{B}(v)$. There are at most $n \cdot |\widehat{B}(v)|$ such edges so the total expected time complexity is

$$\sum_i \sum_{v \in A_i \setminus A_{i+1}} n \cdot \mathbb{E}[|\widehat{B}(v)|] = kn^2 \ell^{1/k} = O(kn^{2+1/k}).$$

The second (and dominating) term of the construction time is to find $\delta(v,\lambda)$ for every $v \in A_{k-1}$ and every $\lambda \in L$. This can either be done by running (the standard version of) Dijkstra's Algorithm from ℓ sources in total $O(m\ell)$ time, or from $|A_{k-1}|$ sources in total $O(m \cdot |A_{k-1}|)$ time. The value $O(m \cdot \min\{\ell, |A_{k-1}|\})$ is maximized when $\ell = |A_{k-1}| = n/\ell^{1-1/k}$ and amounts to $O(mn^{k/(2k-1)})$. This concludes the proof of Theorem 2.

4 Oracles Supporting Dynamic Labels

In this section we consider the situation where the labels may change dynamically. That is, we want a distance oracle that not only supports the usual vertex-label distance queries, but also supports updates of the form $\mathbf{change}(v,\lambda)$, for $v \in V$ and $\lambda \in L$, which sets the label of v to be λ and leaves all labels of vertices in $V \setminus \{v\}$ unchanged. We will show how to modify the oracle scheme of Section 3 so that it supports such queries, at an increase to the stretch and space of the constructed oracles. The main difficulty in achieving this comes from the fact that a vertex may be present in $\Omega(n)$ balls, and thus a change in its label may require updating $\Omega(n)$ hash-tables. The following describes how to overcome this.

4.1 The Data Structure

The construction of our dynamic distance oracle scheme is similar to the construction of Section 3. Below we focus on the main changes. As in Section 3, we

first select the sets of vertices $V = A_0 \supseteq A_1 \supseteq \ldots \supseteq A_{k-1} \supseteq A_k = \emptyset$. However, here we select the vertices in the levels with probability depending on n, as in Section 2. That is, we select A_i by sampling vertices of A_{i-1} independently at random with probability $n^{-1/k}$. Thus, the probability that an arbitrary vertex $v \in V$ is in A_i is exactly $n^{-i/k}$, and the expected-size of A_i is $n^{1-i/k}$. Again, the sets A_i are referred to as the *levels* of G, and we designate for each vertex $v \in V$ a router $r(v) \in V$, defined identically as in Section 3.

The main difference between our static and dynamic schemes lies in the definition of balls. Here our balls will be sets of vertices instead of sets of labels. Moreover, we store half-balls rather than balls. For a vertex $v \in A_i \setminus A_{i+1}$, we define the *half-ball* of v as the set of vertices $B^{\frac{1}{2}}(v) := \{u : \delta(u,v) < \frac{1}{2} \cdot \delta(v, r(v))\}$. That is, a vertex u is in $B^{\frac{1}{2}}(v)$ if it is closer to v than half the distance from v to its router. We also define the cluster $C(v)$ of a vertex $v \in V$ to be the set of all vertices for which v is in their half-ball. That is, $C(v) := \{u : v \in B^{\frac{1}{2}}(u)\}$.

Let us next describe the exact information stored for each vertex $v \in V$. First, we store the cluster $C(v)$ of v, along with all distances $\delta(v,u)$ for $u \in C(v)$. Second, we store all distances to vertices u in the half-ball $B^{\frac{1}{2}}(v)$ of v. The distances to vertices in the half-ball are organized into heaps, one per each label appearing in $B^{\frac{1}{2}}(v)$, that support the following three generic operations:

- **insert**(δ): insert a new distance δ into the heap.
- **remove**(δ): remove an existing distance δ from the heap.
- **minimum**(): return the minimum distance in the heap.

A standard construction of a heap supports the first two operations above in $O(\lg n)$ time (even $O(\lg \lg n)$ time if the edge lengths are integral [10]), while **minimum**() requires constant time. Apart from the cluster $C(v)$, the heaps, the router $r(v)$ and the distance $\delta(v, r(v))$, we also store a hash table at v which allows us to determine in $O(1)$ time whether there exists a vertex labeled λ in $B^{\frac{1}{2}}(v)$, given any label $\lambda \in L$. In Section 4.2 below we show that the expected size of $C(v)$ is $O(kn^{1/k})$ for every vertex $v \in V$. All other information stored at v is asymptotically at most $|B^{\frac{1}{2}}(v)|$, which by similar arguments as those used in Section 3, can also be bounded by $O(kn^{1/k})$ in expectation. Thus, the total size of our data structure can be bounded as in the lemma below.

Lemma 5. *The expected size of our data structure is* $O(kn^{1+1/k})$.

4.2 Label Changes

We next turn to describe how our oracle supports updates of the form **change**(v, λ). The key idea is to use the information stored at the cluster $C(v)$ of v. We begin by bounding the size of $C(v)$ in expectation.

Lemma 6. $\mathbb{E}[|C(v)|] \leq kn^{1/k}$ *for any vertex* $v \in V$.

Proof. For $i \in \{0, \ldots, k-1\}$, let $C_i(v)$ denote the set of vertices $u \in A_i \setminus A_{i+1}$ for which $v \in B^{\frac{1}{2}}(u)$. To prove the lemma, we show that the expected size of

$C_i(v)$ is bounded by $n^{1/k}$. This indeed proves the lemma, since $C(v) = \bigcup_i C_i(v)$, and so by linearity of expectation we get

$$\mathbb{E}[|C(v)|] = \sum_i \mathbb{E}[|C_i(v)|] \leq \sum_i n^{1/k} = kn^{1/k}.$$

For this, let $B_{i+1}(v)$ denote the set of all vertices $u \in A_i \setminus A_{i+1}$ closer to v than A_{i+1}, *i.e.* $B_{i+1}(v) := \{u \in A_i \setminus A_{i+1} : \delta(v,u) < \delta(v, A_{i+1})\}$. We first argue that the expected size of $B_{i+1}(v)$ is at most $n^{1/k}$ using a similar argument as the one used in Lemma 2. Indeed, the size of $B_{i+1}(v)$ can be bounded by the first location of a vertex from A_{i+1} in the list of all vertices in A_i sorted in increasing distance from u. Thus, $\mathbb{E}[|B_{i+1}(v)|]$ is bounded from above by a geometric random variable with rate $n^{-1/k}$, and so $\mathbb{E}[|B_{i+1}(v)|] \leq n^{1/k}$.

To complete the proof of the lemma, we argue that $C_i(v) \subseteq B_i(v)$. Consider an arbitrary vertex $u \in A_i \setminus A_{i+1}$, and suppose that $u \notin B_{i+1}(v)$. Let $w \in V$ be a vertex satisfying $\delta(v,w) = \delta(v, A_{i+1})$. Then $\delta(u, r(u)) \leq \delta(u,w)$, since $w \in A_{i+1}$ and $\delta(u, r(u)) = \delta(u, A_{i+1})$ by definition. Furthermore, as $u \notin B_{i+1}(v)$, we have $\delta(v,w) \leq \delta(v,u)$. Thus,

$$\delta(u, r(u)) \leq \delta(u,w) \leq \delta(u,v) + \delta(v,w) \leq 2\delta(u,v),$$

and so $v \notin B^{\frac{1}{2}}(u)$ by definition. It follows that $u \notin C_i(v)$, and so $C_i(v) \subseteq B_i(v)$. □

The idea behind our label changing procedure is simple: Given an update request of the form **change**(v, λ), we check which half-balls v belongs to using $C(v)$, and update the corresponding heaps at each half-ball. More precisely, for each vertex u for which $v \in B^{\frac{1}{2}}(u)$, we perform a **remove**(δ) operation on the $\lambda(v)$-heap of u with $\delta := \delta(u,v)$, and perform an **insert**(δ) operation on the λ-heap with the same δ. This requires $O(\lg n)$ time for each vertex u with $v \in B^{\frac{1}{2}}(u)$, *i.e.* $u \in C(v)$, since the distance $\delta(u,v)$ is stored in $C(v)$. Thus, according to Lemma 6, we get the following processing time for each label update.

Lemma 7. *The time required for computing* **change**(v, λ) *is* $O(kn^{1/k} \lg n)$.

4.3 Vertex-Label Queries

The query procedure of our oracle is similar to the algorithm in Section 3.2. Again the routers $r_0, \ldots, r_{k-1}$ are defined by $r_0 := v$ and $r_i := r(r_{i-1})$ for $i \in \{1, \ldots, k-1\}$. The algorithm determines the smallest integer $i_0 \in \{0, \ldots, k-1\}$ for which $\lambda \in B^{\frac{1}{2}}(r_{i_0})$, and returns the distance

$$\mathbf{dist}(v, \lambda) := \sum_{0 \leq i < i_0} \delta(r_i, r_{i+1}) + \delta(r_{i_0}, \lambda).$$

The distance $\delta(r_{i_0}, \lambda)$ is retrieved via a single **minimum**() query to the λ-heap stored at $B^{\frac{1}{2}}(r_{i_0})$. The total running-time of our query algorithm is thus $O(k)$. The next lemma bounds the stretch of the output **dist**(v, λ), completing the proof of Theorem 3.

Lemma 8. $\delta(v,\lambda) \leq \mathbf{dist}(v,\lambda) \leq (2 \cdot 3^{k-1} - 1) \cdot \delta(v,\lambda)$ *for any* $(v,\lambda) \in V \times L$.

Proof. Let $(v,\lambda) \in V \times L$, and let $i_0 \in \{0,\ldots,k-1\}$ denote the smallest integer for which $\lambda \in B^{\frac{1}{2}}(r_{i_0})$. We use the following inequality which follows from our definition of the half balls $B^{\frac{1}{2}}(v)$, and from the fact that $\lambda \notin B^{\frac{1}{2}}(r_i)$ for all $i \in \{0,\ldots,i_0-1\}$:

$$\delta(r_i, r_{i+1}) \;\leq\; 2 \cdot \delta(r_i,\lambda) \text{ for all } i \in \{0,\ldots,i_0-1\}. \tag{3}$$

Using induction on i and (3), one can show the following inequality holds in a similar manner as was done in the proof of Lemma 4:

$$\delta(r_i,\lambda) \leq\; 3^i \cdot \delta(v,\lambda) \text{ for all } i \in \{0,\ldots,i_0\}. \tag{4}$$

Thus, we get

$$\begin{aligned}
\mathbf{dist}(v,\lambda) &= \textstyle\sum_{0\leq i<i_0} \delta(r_i, r_{i+1}) + \delta(r_{i_0},\lambda) \\
&\leq \textstyle\sum_{0\leq i<i_0} 2 \cdot \delta(r_i,\lambda) + \delta(r_{i_0},\lambda) \\
&\leq \textstyle\sum_{0\leq i<i_0} 2 \cdot 3^i \cdot \delta(v,\lambda) + 3^{i_0} \cdot \delta(v,\lambda) \\
&= (2 \cdot 3^{i_0} - 1) \cdot \delta(v,\lambda),
\end{aligned}$$

which proves the lemma since $i_0 \leq k-1$. □

This completes the proof of the first part of Theorem 3.

4.4 Small Stretch vs. Efficient Update/Space

We conclude this section by showing that by combining our construction with some of the ideas in [5], it is possible to achieve a stretch-3 oracle with space $O(n^{5/3})$ and label-update time $O(n^{2/3} \lg n)$. Thus, this oracle gives a better stretch than the stretch-5 given by the oracle of Theorem 3, but requires substantially more space and label-update time.

Consider our construction for $k = 2$. Then G has three levels $V := A_0 \supseteq A_1 \supseteq A_2 := \emptyset$, with only A_1 non-trivial. In contrast to our original construction, we consider balls, rather than half-balls, around vertices of G. That is, the set $B(v) := \{u \in V : \delta(v,u) < \delta(v,r(v))\}$ for $v \in V$. Pătraşcu and Roditty [5] show how to randomly select the set A_1 such that the following three properties hold:

- $\mathbb{E}[|A_1|] \leq n^{2/3}$.
- $\mathbb{E}[|B(v)|] \leq n^{1/3}$ for any $v \in A_0 \setminus A_1$.
- $\mathbb{E}[|C(v)|] \leq 2n^{2/3}$ for any $v \in V$.

The first two properties imply that our oracle requires $O(n^{5/3})$ space since we store all $O(n^{4/3})$ distances $\delta(u,v)$ for which $u \in V$ and $v \in A_0 \setminus A_1$, and all $O(n^{5/3})$ distances $\delta(u,v)$ for which $u \in V$ and $v \in A_1$. The last property implies that, as in Section 4.2, we can support label updates for any $v \in V$ in $O(n^{2/3} \lg n)$ time. Moreover, since we store distances in balls, rather than half-balls, the stretch analysis of Lemma 4 applies for this construction. Plugging $k := 2$ in Lemma 4, we get an $O(n^{5/3})$-space oracle with stretch 3 and $O(n^{2/3} \lg n)$ label update time.

5 Discussion

In this paper we defined the natural generalization of distance oracles to vertex-labeled graphs and provided small space and stretch approximate distance oracles. Observe that the known lower bounds [5,7,9] for unlabeled graphs apply for the generalization when all vertices are uniquely labeled. These lower bounds imply that for any fixed $k \geq 1$, any oracle with stretch $2k-1$ requires $\Omega(n^{1+1/k})$ space. Thus, our $O(n\ell^{1/2})$-space oracle with stretch 3 is optimal up to a logarithmic factor. However, we do not know of any better lower bounds, even more so for the dynamic case. Considering this, we list below the important open questions that remain from our work:

1. Is there a vertex-label $(2k-1)$-stretch oracle scheme with $O(kn^{1+1/k})$ space and $O(k)$ query time?
2. Is it possible to get a general scheme of space $O(kn\ell^{1/k})$, with poly(k) stretch and $O(k)$ query time?
3. Can one get a scheme as in (2) that also supports efficient label updates?
4. Are there $O(kn\ell^{1/k})$-space oracles supporting efficient label updates?
5. Are there $O(n^{3/2})$-space oracles with stretch-3 constant-time queries and $O(n^{1/2})$ label updates?

References

1. Baswana, S., Gaur, A., Sen, S., Upadhyay, J.: Distance oracles for unweighted graphs: Breaking the quadratic barrier with constant additive error. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 609–621. Springer, Heidelberg (2008)
2. Baswana, S., Kavitha, T.: Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In: Proc. of the 47th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 591–602 (2006)
3. Erdős, P.: Extremal problems in graph theory. Theory of graphs and its applications, 29–36 (1964)
4. Matoušek, J.: On the distortion required for embedding finite metric spaces into normed spaces. Israel Journal of Mathematics (93), 333–344 (1996)
5. Pătraşcu, M., Roditty, L.: Distance oracles beyond the thorup-zwick bound. In: Proc. of the 51st annual symposium on Foundations Of Computer Science (FOCS), pp. 815–823 (2010)
6. Roditty, L., Thorup, M., Zwick, U.: Deterministic constructions of approximate distance oracles and spanners. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 261–272. Springer, Heidelberg (2005)
7. Sommer, C., Verbin, E., Yu, W.: Distance oracles for sparse graphs. In: Proc. of the 50th IEEE Symposium on Foundation of Computer Science (FOCS), pp. 703–712 (2009)
8. Thorup, M., Zwick, U.: Compact routing schemes. In: Proc. of the 13th ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 1–10 (2001)
9. Thorup, M., Zwick, U.: Approximate distance oracles. Journal of the ACM 52(1), 1–24 (2005)
10. van Emde Boas, P., Kaas, R., Ziljstra, E.: Design and implementation of an effcient priority queue. Mathematical Systems Theory 10, 99–127 (1977)

Asymptotically Optimal Randomized Rumor Spreading

Benjamin Doerr[1,⋆] and Mahmoud Fouz[2]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[2] Faculty of Computer Science, Universität des Saarlandes, Saarbrücken, Germany

Abstract. We propose a new protocol for the fundamental problem of disseminating a piece of information to all members of a group of n players. It builds upon the classical randomized rumor spreading protocol and several extensions. The main achievements are the following:

Our protocol spreads a rumor from one node to all other nodes in the asymptotically optimal time of $(1 + o(1)) \log_2 n$. The whole process can be implemented in a way such that only $O(nf(n))$ calls are made, where $f(n) = \omega(1)$ can be arbitrary.

In spite of these quantities being close to the theoretical optima, the protocol remains relatively robust against failures; for *random* node failures, our algorithm again comes arbitrarily close to the theoretical optima.

The protocol can be extended to also deal with *adversarial* node failures. The price for that is only a constant factor increase in the run-time, where the constant factor depends on the fraction of failing nodes the protocol is supposed to cope with. It can easily be implemented such that only $O(n)$ calls to properly working nodes are made.

In contrast to the push-pull protocol by Karp et al. [FOCS 2000], our algorithm only uses push operations, i.e., only informed nodes take active actions in the network. On the other hand, we discard address-obliviousness. To the best of our knowledge, this is the first randomized push algorithm that achieves an asymptotically optimal running time.

1 Introduction

Broadcasting a piece of information ("rumor") from one node ("source") to all nodes of a network is a classical problem in computer science. The standard model assumes that each node calls at most one node per unit of time. In consequence, if only informed nodes place a call at least $\lceil \log_2 n \rceil$ rounds are needed to spread a rumor to n nodes. Using a broadcast tree that spans the network this bound can be achieved. Such deterministic protocols, however, are vulnerable against failures. In addition, the broadcast tree depends on the source; for each source, each node has to compute or store which neighbors to contact upon receiving the rumor from that source. In consequence, when the network grows, the broadcast tree has to be recomputed.

⋆ Partially supported by the German Science Foundation (DFG) via grant DO 749/4.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 502–513, 2011.

A protocol surprisingly powerful that overcomes these problems is called *randomized rumor spreading*, see, e.g., Feige, Peleg, Raghavan, and Upfal [11], Frieze and Grimmett [13], Karp, Schindelhauer, Shenker, and Vöcking [14]. It proceeds in rounds as follows: in each round, each node that already knows the rumor chooses a communication partner uniformly at random and sends her a copy of this rumor. Thus, each node runs the same randomized process independent of the source node.

In spite of being that simple, this protocol succeeds in spreading the rumor to all nodes of a complete graph in $(1 + o(1))(\log_2 n + \ln n)$ rounds with high probability, i.e., with probability $1-o(1)$. Due to its randomized nature, it is also highly robust against different types of transmission or node failures. Finally, it can handle changes in the size of the network easily.

Feige et al. [11] therefore argue that randomized broadcast algorithms have at least three advantages: simplicity, scalability and robustness.

A clear disadvantage of this simplest version of randomized rumor spreading, however, is the large number of $\Theta(n \log n)$ calls that are necessary. This was overcome in the seminal work by Karp et al. [14]. They present two variations of the randomized rumor spreading protocol which spread the rumor with $O(n \log \log n)$ messages only while still using $O(\log n)$ rounds. Their second protocol is also robust against node failures. A central ingredient are *pull* operations, which allow nodes not yet informed to call random nodes and ask for news. Pull operations, however, have the disadvantage that they create network traffic even if there is no news to be spread. Hence, the assumption underlying the model by Karp et al. [14] is that new rumors are constantly injected into the network.

In this work, we present an alternative solution to the problem. It completely avoids the problematic pull operations. It achieves a broadcast time of $(1 + o(1)) \log_2 n$ and it uses a total number $O(nf(n))$ calls, where $f = \omega(1)$ can be any function tending to infinity arbitrarily slow. This is very close to the theoretically optimal values of $\lceil \log_2 n \rceil$ rounds and $n-1$ calls. Still the protocol is very simple; every node follows the same (randomized) process. Due to its randomized nature, we still have reasonable robustness. For example, if a constant fraction of the nodes chosen uniformly at random crashes at arbitrary times, the time needed to inform all properly working nodes increases by at most a constant factor. Also, if the network size changes, no significant changes are necessary.

The only point in which we assume the protocol to be more powerful compared to previous works is that we discard the address-obliviousness. That is, we assume that each node has a unique label chosen arbitrarily from some ordered set (e.g., the integers). This seems to be reasonable in many settings.

1.1 The Protocol by Karp et al. [14]

As described above, Karp et al. [14] showed how to modify the simple randomized rumor spreading protocol such that instead of $\Theta(n \log n)$ messages only $O(n \log \log n)$ are sufficient to spread a rumor. Roughly speaking, their protocol proceeds as follows. The rumor is equipped with a time stamp (or age counter) in such a way that all nodes that receive the rumor also know for how many rounds

it has been in the network. In each round, each node chooses a random other node as a communication partner. The communication then proceeds in both directions, that is, any partner who knows the rumor forwards it to the other partner. In particular, it is shown that after only $\log_3 n + \Theta(\log \log n)$ rounds of this protocol, all nodes know the rumor with high probability. In addition, a rumor is transmitted in this time interval at most $O(n \log \log n)$ times.

Note that this way of counting ignores all communication effort which does not result in a rumor to be sent. In particular, all calls between two uninformed nodes that arise due to pull operations are not counted. The way this is usually justified is by assuming that there is sufficient traffic in the network due to regular insertions of new rumors. Still, we feel that this is slightly dissatisfying. Note that when using pull operations, there is no way to avoid such communication overhead—a node that did not receive a rumor recently has no way of finding out whether there are rumors around that justify starting pull operations or not. Even nodes that did receive a rumor recently cannot be sure that there is no new rumor that would justify starting pull operations again.

Karp et al. [14] further extend the protocol just sketched to improve its robustness. The one above greatly relies on a very precise estimate of the time when to stop sending on the rumor (here, $\log_3 n + \Theta(\log \log n)$). With transmission failures present, the initial phase of exponential growth might take longer. If this time span is not correctly guessed by the protocol, either nodes remain uninformed, or for too long a time $\Theta(n)$ nodes keep sending out the rumor, leading to too many messages sent. This problem is overcome by a clever median-counting trick. Here, very roughly speaking, nodes average their estimation on how well-known the rumor is with the estimations of their communication partners. This allows the following robustness result. An adversary may specify a set $\mathcal{F}$ of nodes together with arbitrary times at which each of them drops out the game. Nevertheless, within $O(\log n)$ rounds and using $O(n \log \log n)$ messages, all but $O(|\mathcal{F}|)$ nodes are informed. Note that this does allow that up to $O(|\mathcal{F}|)$ 'innocent' nodes (that work properly) remain uninformed.

Karp et al. [14] also prove lower bounds, which show that if in each round all communication is restricted to random matchings of communication partners (i) any address-oblivious algorithm has to make $\Omega(n \log \log n)$ calls and (ii) that any algorithm informing all but a $o(1)$ fraction of the vertices in logarithmic time has to make $\omega(n)$ calls.

1.2 Our Results

The first lower bound stated in the previous paragraph suggests that asking for an address-oblivious protocol may result in only a limited performance being achievable. In addition, one might also wonder if really many broadcasting problems ask for address-oblivious protocols, or if not rather in the majority of settings each participant naturally has a unique address, simply to organize the transport of a message to an addressee.

In this work, we shall drop the requirement of address-obliviousness. However, we shall keep the concept of contacting random neighbors without any

preference, as this seems to be the key to obtaining good broadcasting times, robustness, scalability and small number of calls in all previous works.

Contrary to the model by Karp et al. [14], we do not perform pull operations; all transmissions are initiated by nodes that know the rumor. In consequence, the only direction of informing is from the initiator of the transmission to its addressee, which is chosen uniformly at random, but not always independently.

We do allow, however, two-way communication, in that the addressee acknowledges his readiness to receive the rumor or his knowledge of the rumor. Such a mechanism makes sense anyway, because it allows to reduce the amount of data sent through the network (if the addressee cannot receive the rumor or already knows it, we do not need to send it). In practice, most communication protocols (e.g., the standard network protocol FTP) allow some kind of two-way communication to ensure an error-free transmission.

In this, as we think, natural setting, we propose a protocol that needs only $(1+o(1))\log_2 n$ rounds and $nf(n)$ calls, where $f = \omega(1)$ can be chosen arbitrarily. Note that no protocol that only uses push operations can work in less than $\lceil \log_2 n \rceil$ rounds or using less than $n-1$ calls.

More precisely, we have the following trade-off between rounds and messages. For all $f : \mathbb{N} \to \mathbb{N}$, we give a protocol that needs only $\log_2(n) + f(n) + O(f(n)^{-1} \log n)$ rounds with high probability and $O(nf(n))$ calls. In terms of run-time, this is optimal for $f(n) = \Theta(\sqrt{\log n})$, leading to $\log_2 n + O(\sqrt{\log n})$ rounds and $O(n\sqrt{\log n})$ calls.

The protocol in its basic version is very simple. For the presentation, let us assume that the nodes are numbered from 1 to n, even though what we really need is only that nodes are able (i) to compute the label of a node chosen uniformly at random and (ii) given a label of a node, to compute a uniquely defined successor along a cyclic order of the labels (label plus one, modulo n).

Let $f : \mathbb{N} \to \mathbb{N}$ be given (to formulate the tradeoff scenario). Then the basic protocol works as follows. Each newly informed node sends its first message to another node chosen uniformly at random. From then on, it does the following. If the previous message was sent to a node that was not informed yet, then the next message is sent to the successor of that node in the cyclic order. Otherwise, the next message is sent again to a node chosen uniformly at random. After having encountered $f(n)$ nodes that were already informed, the node stops and does not transmit the rumor anymore. This protocol can be interpreted as a variant of the quasirandom rumor spreading protocol investigated in [5, 6]. In contrast to the latter, all nodes have the same cyclic permutation and can re-start at a random position when they call a node that is already informed.

Despite its simplicity, this protocol shows a good robustness against random node failures. When a randomly chosen fraction of $1-p$ of all nodes fails at arbitrary times, where $p \in (0, 1]$, we still have a running time of $(1+o(1))\log_{1+p} n$ with high probability. Similar arguments as for the $\lceil \log_2 n \rceil$ lower bound in the error-free case show that $\lceil \log_{1+p} n \rceil$ is a valid lower bound on the expected number of rounds needed to distribute the rumor. Thus, we achieve an asymptotically optimal running time even under the presence of *random* node failures.

Similar to the basic protocol by Karp et al. [14], however, this protocol is not very robust against *adversarial* node failures. If an adversary chooses a large consecutive segment of the nodes to be out of order (say ℓ nodes), then there is a reasonable chance (of $\frac{\ell}{2n}$) that the first transmission ever sent hits the first half of this 'failed' segment, and no progress is made for the next $\ell/2$ rounds.

We develop a number of enhancements to cope with such problems. Together, they yield the following. For a given security parameter $p \in (0,1]$, we have a protocol that is robust against adversarial node destruction of up to $(1-p)n$ nodes, i.e., an adversary may destruct arbitrary nodes (excluding the initially informed node) at the beginning of each round such that in total up to $(1-p)n$ nodes are destructed. Such failed nodes do not answer calls directed to them, nor do callers get a feedback if other nodes tried to call a failed node before (this is what causes most of the difficulties). In spite of this strong adversarial setting, we do inform all non-destructed nodes in $(1+\varepsilon)(\log_{1+p} n + \frac{1-p}{p} \ln n)$ rounds for any constant $\varepsilon > 0$, while using $O(n)$ calls to properly-working nodes. The main difficulty in designing such a protocol lies in balancing out the following two effects: on the one hand, a node should not fall back to sending a random message after encountering failed nodes too early, as this would destroy the advantage of following the given order of nodes. On the other hand, in order to be able to cope with long segments of failed nodes, in particular in the early stages of the protocol, such random restarts are necessary.

The main technical difficulty in the analysis of the proposed protocols is that the transmission of messages at each node is not independent, and thus, many classical tools cannot be employed. The key to the solution here is to exploit the existing independence stemming from communications started with random partners. Due to lack of space, most proofs are deferred to the appendix.

In summary, we show that considerable improvements over the fully independent rumor spreading protocol are possible if we do not require the protocol to be address-oblivious. It is thus worth questioning whether the address-obliviousness assumption is really needed in previous applications of the protocol. From the methodological side, our results again show that spicing up randomized algorithms with well-chosen dependencies can yield additional gains. The theoretical analysis might become more complicated, but not so much the algorithm itself.

1.3 Disclaimers

Applications: For reasons of space, we do not give extensive details on randomized rumor spreading and its applications. The seminal papers Feige et al. [11] and Karp et al. [14] contain great discussions of this, better than we could possibly do here. For reports on the actual use of such protocols, see Demers, Greene, Hauser, Irish, Larson, Shenker, Sturgis, Swinehart, and Terry [4] and Kempe, Dobra, and Gehrke [15].

Other network topologies: Randomized rumor spreading can be used on all types of network topologies. Nodes then choose their communication partners at random from the set of their neighbors. For many network topologies,

broadcast times logarithmic in the number of nodes have been shown. Besides the complete graph, they include hypercubes [11], random graphs $G(n,p)$ with $p \geq (1+\varepsilon)\ln(n)/n$ [11] and certain expander graphs [16]. Recently, Chierichetti, Lattanzi, and Panconesi [3] showed that rumor spreading is doable in logarithmic time for graphs of bounded conductance. Doerr, Fouz, and Friedrich [7] showed that a sublogarithmic time can be achieved for social networks modelled by preferential attachment graphs. For Cayley graphs [9] and random geometric graphs [2], the bounds of $O(\mathrm{diam}(G)+\log n)$ are known. In spite of these results, we focussed on the setting where each node has a direct way to communicate with each other node. The reason is that we feel that this is a sufficiently interesting and useful case on its own. Also, of course, it is the setting in which it is easiest to experiment with new ideas. Recall that the concept of reducing the number of messages was also first demonstrated on complete graphs by Karp et al. [14]. Only much later, similar results were obtained for other graph classes [1, 8, 10].

2 The Basic Protocol

Let $G = (V, E)$ be the complete, undirected graph on n nodes. We assume that the nodes of the complete graph are ordered and denote by i the i-th node according to that order. Our goal is to spread a rumor known initially to one node to all nodes in V. We call the node initiating the rumor the *starting node*. A rumor can be transmitted along each edge of the graph in both directions. Every transmission is always initiated by a node that knows the rumor. We count every contact of a node to another node as a *call*. For simplicity, we assume that two nodes never call a node exactly simultaneously even if they both call the same node in the same round; thus, a node is only informed by a single node.

We introduce a simple algorithm that for a certain instantiation achieves, up to lower order terms, an optimal running time. The algorithm is related to the quasirandom protocol by Doerr et al. [5]. There, every node v is equipped with a cyclic permutation $\pi_v : V \to V$ of all nodes in V. Once a node v becomes informed, it chooses one position on its list uniformly at random and contacts the corresponding node in the next round. In each following round, v proceeds according to its permutation π_v, i.e, if v contacted u in the previous round, it now contacts $\pi_v(u)$. Note that different nodes can have different permutations.

Our basic protocol differs from this quasirandom protocol in two main aspects. First, the permutations of all nodes are identical. Second, we introduce the notion of a *restart*: if a node calls an already informed node, it chooses a *random* communication partner in the next round instead of choosing the next one according to the permutation. Each node performs R such restarts, where R is a parameter of the protocol that can be a function of n, and then terminates its rumor spreading. Thus, once a node has called $R+1$ informed nodes, it stops. This rule allows us to bound the total number of calls made. Note that the aspect of keeping the number of calls small was not discussed in [5].

A detailed description of the basic protocol is given in Algorithm 1, where we denote by $j+1$ the successor of j according to the chosen permutation.

The only exception is the starting node. Since there are no other informed nodes yet, there is no need for the starting node to select a random communication partner at the beginning. Thus, it starts informing its own successor node (according to the given permutation) immediately. Only after it encounters the first informed node does it then proceed according to Algorithm 1.

Algorithm 1. Procedure started by newly informed node

```
let R ∈ ℤ⁺ be the number of random calls per node
for i = 1 to R do
    select node j uniformly at random;
    while j not informed // iteration counts as call even if j informed
    do
        inform j; j ← j + 1;
```

2.1 Running Time and Number of Calls

We give an upper bound and an almost matching lower bound on the number of rounds and calls needed by the protocol to spread a rumor from an arbitrary starting node to all nodes of the complete graph.

Theorem 1. *Let $\varepsilon > 0$ be an arbitrarily small constant. With probability $1 - o(1)$, the protocol with R random calls per node informs all nodes in*

$$\begin{aligned} &\log_2(n) + (1+\varepsilon)\ln(n)/R + R + h(n), && \text{if } R \le \sqrt{\ln n} \\ &\log_2(n) + (2+\varepsilon)\sqrt{\ln(n)}, && \text{if } R \ge \sqrt{\ln n} \end{aligned}$$

rounds and $n(R+1)$ calls, where $h(n)$ is a function of arbitrarily slow growth.

Note that by adjusting the stopping parameter R, we get a tradeoff between the number of rounds needed to inform all nodes and the number of calls.

Before analyzing the protocol for general R, we describe two special cases that achieve an (almost) optimal number of rounds and calls, respectively. For $R = \sqrt{\ln n}$, we achieve, up to a lower order term, an optimal running time while using only $O(n\sqrt{\ln n})$ calls. For $R = 1$, we get a very simple broadcasting protocol that, up to constant factors, is both optimal in terms of rounds needed as well as the number of calls.

Corollary 1. *Let $\varepsilon > 0$ be an arbitrarily small constant. With probability $1 - o(1)$, the protocol with*

- *$R = \sqrt{\ln n}$ informs all nodes in $\log_2(n) + (2+\varepsilon)\sqrt{\ln n}$ rounds using $2(1 + 2\varepsilon)n\sqrt{\ln n}$ calls,*
- *$R = 1$ informs all nodes in $\log_2(n) + (1+\varepsilon)\ln n$ rounds using $2n$ calls.*

Before we prove Theorem 1, we make two useful observations for any $R \ge 1$.

Fact 1. *The protocol is always at least as fast as the quasirandom model implemented with identical lists.*

This observation follows from the fact that every node acts as in the quasirandom model until it encounters an informed node. In this case, since we assumed all lists to be the same, the node becomes useless in the quasirandom model as all successive nodes on its list will have also been informed once it tries to call them. In our protocol, however, the node might still call uninformed nodes.

Fact 2. *If a node is* delayed*, i.e., halted for a number of rounds, then the protocol can only become slower.*

To see why this holds, fix for each vertex the R random addressees. Then, a simple induction over time shows that for any set of delays the following holds. No vertex in the delayed model is informed earlier than in the original model. Since this is true for any choice of the R random addressees, the fact follows. This allows us to delay a node for any number of rounds in our analysis.

Proof (of Theorem 1). We distinguish three phases of the process.

The first phase lasts for $\log_2 n + h(n)$ rounds where $h(n)$ is a function that is growing arbitrarily slowly. By Fact 2, we assume that every node is delayed to the second phase once it contacts an informed node. Note that this delayed protocol remains at least as fast as the protocol with $R = 1$ and thus, by Fact 1, also at least as fast as the quasirandom model implemented with identical lists. Fountoulakis and Huber [12] showed that for any arbitrarily small constant $\varepsilon > 0$ the quasirandom model informs $(1-\varepsilon)n$ nodes with probability $1 - o(1)$ in this phase. Thus, we get the same result for our delayed protocol.

The second phase lasts for R rounds. By our delaying assumption, every node that is informed in the first phase will remain active for at least $R - 1$ rounds before the second phase ends. The crucial observation is that every informed node that is still active either informs an uninformed node in a single round or calls a random node in the next round. The former happens at most εn times in total. We conclude that at the end of the second phase the number of random calls made is at least $(1-\varepsilon)nR - \varepsilon n \geq (1-2\varepsilon)nR$ (including the random calls made in the first phase). We use this to bound the largest interval of uninformed nodes by $(1+3\varepsilon)\ln(n)/R$.

Let I be an interval of length $(1+3\varepsilon)\ln(n)/R$. Then, the probability that no node in I becomes informed in the second phase by these random calls is at most $\left(1 - \frac{(1+3\varepsilon)\ln n}{nR}\right)^{(1-2\varepsilon)nR} \leq \exp\left(-(1-2\varepsilon)(1+3\varepsilon)\ln n\right) = n^{-1-\varepsilon+6\varepsilon^2} = n^{-1-\varepsilon'}$, for some constant $\varepsilon' > 0$ (when ε is sufficiently small). By a union bound argument, we conclude that there is no completely uninformed interval of length $(1+3\varepsilon)\ln(n)/R$ after the second phase with probability at least $1 - n^{-\varepsilon'}$ for some constant $\varepsilon' > 0$.

In the last phase, all the remaining uninformed intervals are 'processed'. This takes at most the length of the largest uninformed interval, which is at most $(1+3\varepsilon)\ln(n)/R$. Note that here we exploit the exceptional behavior of the starting node at the beginning; if the starting node were to choose a random communication partner from the beginning, there could be an uninformed interval on the cyclic list after the starting node that is not further processed.

Using a simple union bound, we bound the total failure probability by $o(1)$.

It remains to bound the number of calls. Note that each node calls at most R informed nodes in total. Hence, we use at most n calls to inform all nodes and, in addition, at most nR calls until all nodes stop informing.

We can also show that the upper bound is essentially sharp.

Theorem 2. *Let $\varepsilon > 0$. If the protocol with R random calls per node is run for less than*

$$\begin{aligned} \log_2(n) + (1-\varepsilon)\ln(n)/R + \tfrac{1}{2}R, &\qquad \text{if } R \le \sqrt{2(1-\varepsilon)\ln n}, \\ \log_2(n) + \sqrt{2(1-\varepsilon)\ln n}, &\qquad \text{if } R \ge \sqrt{2(1-\varepsilon)\ln n} \end{aligned}$$

rounds, then with probability $1 - \exp(-n^{\Theta(\varepsilon)})$ not all nodes are informed.

2.2 Robustness against Random Node Failures

Despite its simplicity, our protocol offers reasonable robustness. We consider the following natural node failure model: each node apart from the starting node independently sampled with probability $p \in (0, 1]$ works properly. Nodes that do not work properly are called *failed*. These nodes may stop answering calls or sending out messages at arbitrary times (specified by an adversary). If a node contacts one that has stopped working, it does not get a feedback and continues with the successor of the failed node in the next round (hence a failed node does not pretend to be informed). Not surprisingly, we cannot hope to achieve robustness in such a situation without any sacrifices. For example, when we are confronted with a linear number of randomly distributed failed nodes, it is unreasonable to assume that it is possible to inform all properly working nodes in $(1 + o(1)) \log_2 n$ rounds. It is easy to see that any such algorithm needs, in expectation, at least $(1 - o(1)) \log_{1+p} n$ rounds to inform all properly working nodes. It turns out that in this case the basic protocol can be instantiated to have a running time of at most $(1 + o(1)) \log_{1+p} n$.

Lemma 1. *If nodes fail independently with probability $1 - p \in (0, 1]$, then all properly working nodes are informed in*

$$\begin{aligned} (1+o(1))(\log_{1+p} n + \tfrac{1}{Rp^2}\ln n) + R, &\qquad \text{if } R < \tfrac{1}{p}\sqrt{\ln n}, \\ (1+o(1))(\log_{1+p} n + \tfrac{2}{p}\sqrt{\ln n}), &\qquad \text{if } R \ge \tfrac{1}{p}\sqrt{\ln n} \end{aligned}$$

rounds using $(1 + o(1))(R + 1)n/p$ calls, with probability $1 - o(1)$.

3 Making the Protocol Robust against an Adversary

The main disadvantage of the basic protocol is its lack of robustness in the face of general, *non-random* transmission failures. Assume for example that a large segment of nodes that come consecutively in the permutation is not working.

Then there is a reasonable chance that the first transmission sent by the starting node ends up in this faulty segment and hence the protocol needs time linear in the length of that segment to get out of it again. Thus, if we assume to have a linear number of such failed nodes, then with constant probability the protocol will take a linear number of steps until all nodes are informed.

We now describe a modification of the algorithm that overcomes this problem, still achieves a logarithmic running time and performs a linear number of calls to properly working nodes. We assume that an adversary specifies a set of failed nodes. In contrast to the previously studied model, we distinguish between a *successful* call, i.e., a call to a properly working node, and an *unsuccessful* call, i.e., a call to a failed node.

This setting turns out to be much more difficult to analyse. Even the more complex *median-counter algorithm* by Karp et al. [14], which relies on pull operations by uninformed nodes, only achieves a running time of $\Theta(\ln n)$ while using $\Theta(n \ln \ln n)$ calls. Moreover, their algorithm only guarantees to inform all but $O(|F|)$ nodes, where F is the set of failed nodes.

On the other hand, our algorithm informs *all* properly working nodes in $O(\ln n)$ rounds, relies only on push operations, needs only $O(n)$ successful calls and overall $O(n \ln n)$ calls. The reason why the total number of calls can be much larger than the number of successful calls lies in the end phase of the protocol. Just as in the basic protocol, the end phase is characterized by having informed almost all properly working nodes. An adversary could have distributed the remaining properly working nodes in a large bunch of such failed nodes. Since an informed node that is processing the list in such a segment of failed nodes cannot decide whether another node already passed by these nodes, we could end up spending a linear number of useless calls per node. To bound the number of calls, we introduce a security parameter p that the user can set for the fraction of properly working nodes. As long as p is a valid lower bound, we guarantee that at most $\frac{1+\varepsilon}{p} n \ln(n)\big(2 + \log_p(\varepsilon/12)\big)$ calls are made for any constant $\varepsilon > 0$.

The protocol is very similar to the basic protocol. When a node receives the rumor, it starts the procedure described in Algorithm 2. As before the starting node begins by selecting its own successor node instead of a random node. It then proceeds just as in Algorithm 2, i.e., we only replace its first random selection by its successor node. The main difference to the basic protocol is that a newly informed node first performs random calls until it has found a properly working node. It then proceeds just as in the basic protocol. Intuitively, this modification prevents the bad scenario described above where the first call goes to a large segment of failed nodes and is then stuck there for a long time. For the stopping rule, we introduce three counters for each node. All these counters log the behavior of the node after it found a properly working node.

- The *restart counter* of a node counts how many *informed* nodes it has called in total.
- The *global failure counter* of a node counts how many *failed* nodes it has called in total.

Algorithm 2. Procedure started by newly informed node

```
r ← 0, lfc ← 0, gfc ← 0, R ← log_p(ε/12), L ← (1+ε) ln(n)/p,
G ← (1+ε) ln(n)/p;
repeat
    select node j uniformly at random;
until j properly working node ;
while r < R and gfc < G do
    while j not informed and lfc < L do
        if j failed then
            lfc ← lfc + 1; gfc ← gfc + 1;
        else
            inform j; j ← j + 1; lfc ← 0;
    r ← r + 1; select node j uniformly at random;
```

– The *local failure counter* of a node i counts the number of failed nodes i has encountered in a row since the last random call of i.

The stopping rule is as follows. A node stops informing immediately if either the restart counter reaches $R = \log_p(\varepsilon/12)$ or the local failure counter reaches $L = (1+\varepsilon)\ln(n)/p$. Furthermore, if the global failure counter reaches $G = (1+\varepsilon)\ln(n)/p$, then the node stops informing the next time it calls an informed node. Whereas the restart counter fulfills the same role as in the basic protocol, the global failure counter makes sure that not too many calls are wasted on failed nodes. The local failure counter ensures that all properly working nodes are informed, even those that might be 'hidden' in a segment of failed nodes.

Theorem 3. *Let $\varepsilon \in (0,1)$ and $p \le (n-|F|)/n$, where F is the set of failed nodes in a graph. Then, the protocol with a random start-up phase informs, with probability $1-O(n^{-p\varepsilon/32})$, all properly working nodes in at most $(1+\varepsilon)\big(\log_{1+p} n + \frac{1-p}{p}\ln n\big)$ rounds using $O(n)$ successful calls and $\frac{1+\varepsilon}{p} n \ln n + O(n)$ calls in total.*

Acknowledgments

We thank Carola Winzen for valuable comments on the manuscript.

References

[1] Berenbrink, P., Elsässer, R., Friedetzky, T.: Efficient randomized broadcasting in random regular networks with applications in peer-to-peer systems. In: Proc. of 27th ACM Symposium on Principles of Distributed Computation (PODC), pp. 155–164 (2008)

[2] Bradonjic, M., Elsässer, R., Friedrich, T., Sauerwald, T., Stauffer, A.: Efficient broadcast on random geometric graphs. In: Proc. of the 21st Annual ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 1412–1421 (2010)

[3] Chierichetti, F., Lattanzi, S., Panconesi, A.: Almost tight bounds for rumour spreading with conductance. In: Proceedings of the 42th ACM Symposium on Theory of Computing (STOC), pp. 399–408 (2010)

[4] Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: Proceedings of the 6th ACM Symposium on Principles of Distributed Computing (PODC), pp. 1–12 (1987)
[5] Doerr, B., Friedrich, T., Sauerwald, T.: Quasirandom rumor spreading. In: Proc. of the 19th ACM-SIAM Symp. on Disc. Alg. (SODA), pp. 773–781 (2008)
[6] Doerr, B., Friedrich, T., Sauerwald, T.: Quasirandom rumor spreading: Expanders, push vs. Pull, and robustness. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 366–377. Springer, Heidelberg (2009)
[7] Doerr, B., Fouz, M., Friedrich, T.: Social networks spread rumors in sublogarithmic time. In: To appear in the Proceedings of the 43th ACM Symposium on Theory of Computing, STOC (2011)
[8] Elsässer, R.: On the Communication Complexity of Randomized Broadcasting in Random-like Graphs. In: Proceedings of the 18th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 148–157 (2006)
[9] Elsässer, R., Sauerwald, T.: Broadcasting vs. Mixing and information dissemination on cayley graphs. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 163–174. Springer, Heidelberg (2007)
[10] Elsässer, R., Sauerwald, T.: On the Power of Memory in Randomized Broadcasting. In: Proceedings of the 19th ACM-SIAM Symp. on Disc. Alg. (SODA), pp. 218–227 (2008)
[11] Feige, U., Peleg, D., Raghavan, P., Upfal, E.: Randomized broadcast in networks. Random Structures and Algorithms 1, 447–460 (1990)
[12] Fountoulakis, N., Huber, A.: Quasirandom rumor spreading on the complete graph is as fast as randomized rumor spreading. SIAM Journal on Discrete Mathematics 23, 1964–1991 (2009)
[13] Frieze, A., Grimmett, G.: The shortest-path problem for graphs with random arc-lengths. Discrete Applied Mathematics 10, 57–77 (1985)
[14] Karp, R., Schindelhauer, C., Shenker, S., Vöcking, B.: Randomized Rumor Spreading. In: Proceedings of the 41st IEEE Symposium on Foundations of Computer Science (FOCS), pp. 565–574 (2000)
[15] Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS), pp. 482–491 (2003)
[16] Sauerwald, T.: On mixing and edge expansion properties in randomized broadcasting. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 196–207. Springer, Heidelberg (2007)

Fast Convergence for Consensus in Dynamic Networks

T.-H. Hubert Chan and Li Ning

The University of Hong Kong

Abstract We study the convergence time required to achieve consensus in dynamic networks. In each time step, a node's value is updated to some weighted average of its neighbors' and its old values. We study the case when the underlying network is dynamic, and investigate different averaging models. Both our analysis and experiments show that dynamic networks exhibit fast convergence behavior, even under very mild connectivity assumptions.

1 Introduction

Natural group behavior is exhibited in many dynamic systems. Typically, each individual or node in the set V has some number in $\mathbb{R}$, which can represent one's opinion. In every time step, an individual observes the opinions of a subset of other individuals and updates one's opinions accordingly. It is observed that in many such systems[17,18,6], the values of all nodes converge to the same value (or opinions of individuals reach consensus) after a small number of iterations, even though each node only interacts with a small number of other nodes in each time step.

The *weighted averaging model* [7], used by DeGroot to model consensus of opinions, has been widely studied to explain convergent behavior in such networks. The value $v_t[i]$ of an individual at time step t is updated by taking some weighted average of all individuals' values: $v_{t+1}[i] := \sum_j p_t[i,j] \cdot v_t[j]$, where each $p_t[i,j]$ is non-negative and $\sum_j p_t[i,j] = 1$. Typically, for each i, there is only a small number of j's such that $p_t[i,j]$ is non-zero; those correspond to the individuals whose values can be observed by i. The interactions of individuals in a time step can be represented by a network $G_t = (V, E_t)$, where an edge $\{i,j\} \in E_t$ means the individuals can observe each other's values at time t. Besides its simplicity, the weighted averaging model has applications in parallel computation [1], control theory [9,3,2,8,5,12] and ad hoc networks [11].

In this paper, we study what weighting strategies and what kind of networks can enable fast convergence to achieve consensus. In particular, for different weighting strategies and network properties, we analyze the number of time steps that is sufficient for all nodes' values to be close to one another. The *uniform averaging model* is the case when given a network G_t, an individual updates its value to the average of its neighbors' and its old values. We consider the case when the underlying network topology is dynamic, i.e. the networks G_t's change

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 514–525, 2011.

over time. To keep our analysis as general as possible, we do not specify how the networks G_t's evolve (which may or may not depend on the nodes' values); we only assume general structural properties of the networks such as degree distribution and connectivity.

Related Work. The special case of the uniform averaging model with time-invariant network topology is well-understood [7]. Using the theory of stochastic matrices and spectral graph theory, it is known that the convergence time is related to the eigenvalue gap [10] of the transition matrix P involved. If the underlying network is time-invariant and connected, Olshevsky and Tsitsiklis [16] showed that the convergence time for the uniform averaging model is $O(n^3)$.

Relatively little is known about the convergence time when the underlying network is dynamic. Assuming some special structure in the network in each time step, Cao et. al [3] showed that the convergence time is $n^{O(n)}$. Olshevsky and Tsitsiklis [16] also considered weak connectivity assumptions: in the given sequence, the union of any k consecutive networks is connected. In this case, they showed that the convergence time under the uniform averaging model is $O(kn^{kn})$ and a lower bound of $\Omega(n)^k$. Using a "load balancing" algorithm, they can achieve $O(n^3)$ convergence time. They also showed a convergence time of $O(n^3)$ for the uniform averaging model with the fixed degree assumption [15]. With the same weak connectivity assumption, Netić et. al [14] showed convergence time of $O(\frac{kn^2}{\alpha})$, for the special case where the transition matrices are doubly stochastic and $\alpha > 0$ is a a lower bound on the non-zero entries.

Vicsek et. al [19] used the weighted averaging model to study interaction between particles, which influence one another's velocities. Two particles can influence each other if their distance is close enough. The system reaches a convergent state when all particles are traveling in nearly the same direction. Jadbabaie et. al [9] gave a theoretical explanation to such convergent behavior. Recently, Chazelle [4] considered a discrete version of the model and showed that the convergence time is $O(2 \uparrow\uparrow n)$.

Other interaction models have also been studied. In [2], directed networks and asynchronous updates were considered. In [8,5,12], convergence under non-linear update rules were studied.

Our Contribution and Results. In this paper, we give a quantitative analysis between the convergence time and the connectivity of the networks in the given sequence. If convergent behavior is observed at all in real systems, then the number of time steps taken certainly cannot be $O(n^n)$ or even $O(2 \uparrow\uparrow n)$. Typically, the networks concerned are well-connected and convergence time of $O(\log n)$ is observed.

For static network, this can be easily explained by the theory of stochastic matrices and spectral graph theory. The update process in each time step corresponds to multiplication by a stochastic matrix P. Although P is in general not symmetric (and hence the eigenvectors are not mutually orthogonal), for all positive integers t, the powers P^t all have the same eigenvectors, and any eigenvalue gap in P will be magnified in P^t. However, if the underlying network is dynamic,

then the corresponding transition matrices will not have the same eigenvectors anymore (apart from the all one's vector), and hence the above argument does not work.

In Section 3, we overcome this technical hurdle by choosing the weights carefully such that in the transformed space the eigenvectors are mutually orthogonal. Assuming that each node has limited degree variation in the given network sequence, and each network is well-connected (as measured by conductance), we can obtain an eigenvalue gap in the transition matrix in each time step. Combining these techniques, we show that convergence time is $O(\log n)$. If we just assume that each network is well-connected (without the assumption on limited degree variation), we have $O(n)$ convergence time. As far as we know, the previous best known convergence time under any weighted averaging model for dynamic connected networks is $O(n^3)$ [14].

Under the uniform averaging model, we analyze in Section 4 the conditions on the given network sequence such that fast convergence can be obtained. Assuming that each network is degree bounded and for some integer k, the union of every k consecutive networks is a vertex expander, we show that the convergence time is polynomial by using the expansion property directly. Furthermore, our techniques can be extended to the probabilistic case where the connectivity condition for each union of k networks only needs to hold with some positive probability.

On the other hand, our simulations in Section 5 show that for well-connected graphs such as $G_{n,p}$, the convergence time under the uniform averaging model grows logarithmically with the network size, suggesting that there is a lot of room for improvement. It would be an interesting open problem to determine the most general conditions on the networks under which the uniform averaging model has fast convergence time.

2 Preliminary

Suppose there is a set V of n individuals and each one of them holds an opinion which can be represented by a number from $\mathbb{R}$. An *opinion configuration* at some time t is an n-dimensional vector from $\mathbb{R}^n$. We denote the configuration at time t by v_t, and the opinion of individual i by $v_t[i]$.

At each time step, the individuals form a network (in this document we use the terms "network" or "undirected graph" interchangeably) $G_t = (V(G_t), E(G_t))$, in which the nodes represent the individuals and an edge between two nodes means they can potentially communicate their opinions to each other. We assume all G_t's have the same set of nodes, i.e. $V(G_t) = V$ for all t. Moreover, we assume that the sequence $\{G_t\}$ of networks is generated by some process that is, in general, independent of the individuals' opinions v_t's.

We use the maximum difference between two individual's numbers to measure how close a configuration reaches consensus.

Definition 1 (τ-Measure). *Given a configuration vector $v \in R^n$, the τ-measure of v is $\tau(v) = \max_{i,j} |v[i] - v[j]|$.*

We say that the vector v achieves consensus when $\tau(v) = 0$; and for $\epsilon > 0$, the vector v achieves ϵ-consensus when $\tau(v) \leq \epsilon$.

We next describe the models we use to analyze the convergent behavior for dynamic systems.

2.1 Convergence Model for Dynamic Networks

Given a sequence $\{G_t : t \geq 0\}$ of networks and some initial configuration $v_0 \in \mathbb{R}^n$, we describe the update rule for each time step. At time step t, the nodes are connected by the network G_t; we denote the degree of node i by $d_t[i]$. Moreover, each node i has some positive integral weight $w_t[i] \geq d_t[i]+1$, which indicates how resistant the individual is to others' opinions, with a higher weight indicating higher resistance. The update rule for each node i at time t is given by the following equation.

$$v_{t+1}[i] = (1 - \frac{d_t[i]}{w_t[i]}) \cdot v_t[i] + \frac{1}{w_t[i]} \sum_{j:\{i,j\} \in E(G_t)} v_t[j]. \tag{1}$$

Matrix Notation. The update rule can be expressed succinctly using matrix notation. We treat w_t, v_t, and d_t as $n \times 1$ column vectors. Given a network G_t, recall its Laplacian L_t is defined as the $n \times n$ matrix such that $L_t[i,j]$ is $d_t[i]$ when $i = j$, -1 when $\{i,j\} \in E(G_t)$, and 0 otherwise.

Given a square matrix A, we denote its trace by $\mathsf{tr}(A)$, i.e., the sum of its diagonal entries. Given a vector $w \in \mathbb{R}^n$, we use $\mathsf{Diag}(w)$ to denote the diagonal matrix such that $\mathsf{Diag}(w)[i,i] = w[i]$. Given a weight vector $w_t \in \mathbb{R}^n$, let $W_t = \mathsf{Diag}(w_t)$ and the transition matrix $P_t = I_n - W_t^{-1} L_t$, where I_n is the $n \times n$ identity matrix. Then, equation (1) can be rewritten as:

$$v_{t+1} = P_t v_t. \tag{2}$$

Special Cases. We describe some special cases for the weights w_t.

- **Static Weight Model.** In this case, there is some fixed weight vector $w \in \mathbb{R}^n$ such that for all time steps t, $w_t = w$. Observe that in this case, we need to restrict the networks such that for all t and all nodes i, the degree $d_t[i] \leq w[i] - 1$. To ensure that each node is still influenced by its neighbors, we normally also assume $w[i] = O(n)$ for all t.
- **Uniform Averaging Model.** In this case, for each time step t and each node i, $w_t[i] = d_t[i] + 1$. Observe that in this case, the new opinion of a node is simply the average of the sum of its and its neighbors' opinions. Hence, equation (1) reduces to $v_{t+1}[i] = \frac{1}{d_t[i]+1}(v_t[i] + \sum_{j:\{i,j\} \in E(G_t)} v_t[j])$.

Convergence Time. Given some initial configuration $v_0 \in \mathbb{R}^n$, and some convergence process, for $\epsilon > 0$, the *convergence time* to achieve ϵ-consensus is the minimum T such that for all $t \geq T$, $\tau(v_t) \leq \epsilon$.

Union Network. We do not always require each network G_t to be connected. We can still prove convergence results as long as the union of the networks over a certain period of time is well-connected. Formally, suppose I is a set of time indices for the collection of networks $\{G_t = (V, E_t) : t \in I\}$. Then, the union network is defined as $\cup_{t\in I} G_t := (V, \cup_{t\in I} E_t)$.

2.2 Stochastic Matrices

We mention some useful results about *stochastic matrices*. Recall that an $n \times n$ matrix M is *row stochastic* (or simply *stochastic*) if all its entries are non-negative and the entries of each row sum to 1. Observe that the transition matrix P_t in (2) is stochastic. Recall that the product of two stochastic matrices is still stochastic. We define two measures for matrices, which describe how different the rows of a matrix are.

Definition 2 (τ_1- and τ_2-Measures). *Given a matrix P, the τ_1-measure of P is defined as $\tau_1(P) = \frac{1}{2}\max_{i,j}\{\sum_k |P[i,k] - P[j,k]|\}$; the τ_2-measure of P is defined as $\tau_2(P) = \max_{i,j}\{(\sum_k |P[i,k] - P[j,k]|^2)^{\frac{1}{2}}\}$.*

Observe that for a column vector v, $\tau(v) = 2\tau_1(v) = \tau_2(v)$.

Fact 1 states an important relationship between the τ_2-measure of the product of two matrices and product of the measures of the corresponding matrices. Its proof is given in [4,13]. Fact 2 relates the τ_1-measure of a stochastic matrix with its smallest entry.

Fact 1. *For any stochastic matrix A and any matrix B, whose dimensions are compatible with A such that AB is well-defined, we have $\tau_2(AB) \le \tau_1(A)\tau_2(B)$.*

Observe that any stochastic matrix P has the property that $\tau_1(P) \le 1$. Hence, it follows that for all t, $\tau(v_{t+1}) = \tau_2(P_t v_t) \le \tau(v_t)$.

Fact 2. *Suppose P is a stochastic matrix such that all its entries are at least some number $\alpha > 0$. Then, $\tau_1(P) \le 1 - n\alpha$.*

3 Static Weight Model: Limited Degree Variation and Well-Connected Dynamic Networks

In this section, we show that fast convergence for the static weight model is achieved if in the given sequence $\{G_t\}$ of networks, each G_t is *well-connected*, and for each node i, the degree $d_t[i]$ does not vary too much with respect to t. In particular, we explore a quantitative relationship between the convergence time and the connectivity of the given networks.

The concept *conductance* can be used to measure how connected a graph is.

Definition 3 (Conductance). *Given a network $G = (V, E)$, and a subset $S \in V$, the edge border set of S is defined as $\partial(S) = \{\{u, v\} \in E | u \in S, v \in \overline{S} = V \setminus S\}$. The conductance of G is defined as $\Psi(G) = \sup\{\varphi > 0 | \frac{\partial(S)}{\min\{d(S), d(\overline{S})\}} \ge \varphi, \forall S \subset V\}$, where $d(S) = \sum_{i\in S} d[i]$.*

Given any stochastic matrix P, we sort and label its eigenvalues in the descending order of the eigenvalues' magnitude, i.e., $|\lambda_0(P)| \geq |\lambda_1(P)| \geq \cdots \geq |\lambda_{n-1}(P)|$, where $\lambda_0(P) = 1$. The following lemma, which is an extension of the Cheeger's Inequality, relates the spectral properties of a transition matrix P and the conductance of the underlying network. We defer its proof to the full version.

Lemma 1 (Conductance Implies Eigenvalue Gap). *Suppose G is a network with conductance Ψ, and $w \in \mathbb{Z}^n$ is a positive weight vector such that for each i, its degree $d[i]$ satisfies $2d[i] \leq w[i] \leq K \cdot d[i]$. Define the transition matrix $P := I_n - W^{-1}L$, where $W = \mathsf{Diag}(w)$ and L is the Laplacian of G. Then, $|\lambda_1(P)| \leq 1 - \eta$, where $\eta = \frac{\Psi^2}{2K}$.*

Theorem 1 (Static Weight Model). *Given a positive weight vector $w \in \mathbb{Z}^n$, and a sequence $\{G_t : t \geq 0\}$ of networks, let the transition matrix $P_t := I_n - W^{-1}L_t$, where $W = \mathsf{Diag}(w)$ and L_t is the Laplacian of G_t. Suppose there is some $0 < \eta < 1$ such that for all t, $|\lambda_1(P_t)| \leq 1 - \eta$. Then, for any initial configuration vector v_0, the convergence time to achieve ϵ-consensus is $O(\frac{1}{\eta} \log \frac{\|W^{\frac{1}{2}} v_0\|_2}{\epsilon})$, which is $O(\frac{1}{\eta} \log \frac{n\|v_0\|_2}{\epsilon})$, if for each i, $w[i] = O(n)$. For the special case when all nodes i have the same $w[i]$, the convergence time can be improved to $O(\frac{1}{\eta} \log \frac{\|v_0\|_2}{\epsilon})$.*

Proof. Let $\lambda := 1 - \eta$. Suppose $P := I_n - W^{-1}L$ is the transition matrix corresponding to some network with Laplacian L such that $|\lambda_1(P)| \leq \lambda$. Consider $M := W^{\frac{1}{2}} P W^{-\frac{1}{2}}$ and observe that M is symmetric and has exactly the same eigenvalues as P. In particular, M has eigenvalue 1 with the corresponding eigenvector $u_0 = \mathsf{tr}(W)^{-\frac{1}{2}} W^{\frac{1}{2}} \mathbf{1}$, where $\mathbf{1}$ is the all one's vector. Moreover, since the eigenvectors of a symmetric matrix are mutually orthogonal, we have for each vector z that is orthogonal to u_0, the vector Mz is still orthogonal to u_0 and $||Mz||_2 \leq \lambda ||z||_2$. For each t, we define $M_t := W^{\frac{1}{2}} P_t W^{-\frac{1}{2}}$.

Given an initial configuration vector v_0, we write $W^{\frac{1}{2}} v_0 = x + y$, where x is parallel to u_0 and y is orthogonal to u_0. According to our convergence model, we have

$$v_t = P_{t-1} \cdots P_0 v_0 = W^{-\frac{1}{2}} M_{t-1} \cdots M_0 W^{\frac{1}{2}} v_0 = W^{-\frac{1}{2}} x + W^{-\frac{1}{2}} M_{t-1} \cdots M_0 y.$$

We next observe that all entries of $W^{-\frac{1}{2}} x$ are identical, and hence $\tau(v_t) = \tau(W^{-\frac{1}{2}} M_{t-1} \cdots M_0 y)$, which is at most $2||W^{-\frac{1}{2}} M_{t-1} \cdots M_0 y||_2$, because for any vector v, $\tau(v) \leq 2||v||_2$.

Observing that $W^{-\frac{1}{2}}$ is a diagonal matrix such that each entry is at most 1, we have $||W^{-\frac{1}{2}} M_{t-1} \cdots M_0 y||_2 \leq ||M_{t-1} \cdots M_0 y||_2 \leq \lambda^t ||y||_2 \leq \lambda^t ||W^{\frac{1}{2}} v_0||_2$, where the last inequality holds because x and y are orthogonal, and the penultimate inequality holds because for each $0 \leq k \leq t$, the vector $M_{k-1} \cdots M_0 y$ remains orthogonal to u_0, and hence is spanned by eigenvectors (of each M_i) whose eigenvalues have absolute values at most λ.

Hence,we have $\tau(v_t) \leq 2\lambda^t \|W^{\frac{1}{2}} v_0\|_2$, which is at most ϵ, for $t = \Omega(\frac{1}{\eta} \log \frac{\|W^{\frac{1}{2}} v_0\|_2}{\epsilon})$, where $\eta = 1 - \lambda$.

Finally, observing that for the special case when for all i, $w[i] = \omega$ is the same, we have $W = \omega I_n$. Hence, in the above argument the $W^{\frac{1}{2}}$ and $W^{-\frac{1}{2}}$ cancel with each other, and we can conclude instead that $\tau(v_t) \leq 2\lambda^t \|v_0\|_2$, and so the convergence time becomes $O(\frac{1}{\eta} \log \frac{\|v_0\|_2}{\epsilon})$. □

Hence, from Lemma 1 and Theorem 1, we have the following corollary.

Corollary 1 (Logarithmic Convergence Time for Limited Degree Variation). *Given a positive weight vector $w \in \mathbb{Z}^n$, and a sequence $\{G_t : t \geq 0\}$ of networks, let the transition matrix $P_t := I_n - W^{-1}L_t$, where $W = \mathsf{Diag}(w)$ and L_t is the Laplacian of G_t. Suppose there is some $\Psi > 0$ and some $K > 0$, such that for all t, G_t has conductance at least Ψ and for each node i, its degree $d_t[i]$ satisfies $2d_t[i] \leq w[i] \leq Kd_t[i]$. Then, for any initial configuration vector v_0, the convergence time to achieve ϵ-consensus is $O(\frac{K}{\Psi^2} \log \frac{n\|v_0\|_2}{\epsilon})$, if for each i, $w[i] = O(n)$.*

For the special case when $w[i] = 2n$ for all nodes i, we have the following corollary, also using Lemma 1 and the identical weight case in Theorem 1.

Corollary 2 (Linear Convergence Time). *Given a sequence $\{G_t : t \geq 0\}$, suppose that there is some $\Psi > 0$ such that for all t, G_t has conductance at least Ψ. We set the weight vector $w \in \mathbb{Z}^n$ such that for all i, $w[i] = 2n$, and consider the transition matrix $P_t := I_n - W^{-1}L_t$ as before. Then, given $\epsilon > 0$ and initial configuration vector $v_0 \in \mathbb{R}^n$, the convergence time to achieve ϵ-consensus is $O(\frac{n}{\Psi^2} \log \frac{\|v_0\|_2}{\epsilon})$.*

4 Analysis of the Uniform Averaging Model

In this section, we analyze the convergence time for the uniform averaging model. Given a network G_t, we consider the weight vector w_t such that $w_t[i] = d_t[i] + 1$, the degree of node i plus 1. The transition matrix is given by $P_t = I_n - \mathsf{Diag}(w_t)^{-1}L_t$, where L_t is the Laplacian of G_t.

We also assume that each network in the sequence $\{G_t\}$ has degree bounded by some d, i.e., for all t and all i, $d_t[i] \leq d$. However, we only need weak connectivity assumptions on the given sequence of networks. We do not even require each network to be connected. All we need is that there is some integer k such that the union of the networks in every k consecutive time steps is well-connected. Although we only prove convergence time polynomial in n, experiments in Section 5 suggest that the convergence time for the uniform averaging model is $O(\log n)$ for well-connected networks.

4.1 Weakly Connected Networks

Given a network, the standard notion of vertex expansion can measure its connectivity.

Definition 4 (Vertex Expansion). *Given a network (undirected graph) $G = (V, E)$ and a subset $S \subseteq V$, the vertex border set of S is defined as $\delta(S) = \{v \in V(G) \setminus V(S) | \exists u \in S, s.t. \{u, v\} \in E\}$. The vertex expansion of G is defined as $\Phi(G) = \sup\{\phi > 0 | \frac{\delta(S)}{|S|} \geq \phi, \forall S \subset G, |S| \leq \frac{|V|}{2}\}$.*

Definition 5 (Union Vertex Expansion). *Given a sequence $\{G_t : t \geq 0\}$ of networks, and an integer $k \geq 1$, we say the sequence has k-union vertex expansion at least ϕ if for any $t \geq 0$, $\Phi(\cup_{j=t}^{t+k-1} G_j) \geq \phi$.*

The main result of this section is given in the following theorem, which is a direct consequence of Lemmas 2 and 3

Theorem 2 (Convergence Time for Union Vertex Expanders). *Suppose the network sequence $\{G_t\}$ with bounded degree d has k-union vertex expansion at least $\phi > 0$. Then, given an initial vector $v_0 \in \mathbb{R}^n$ and $\epsilon > 0$, the convergence time to achieve ϵ-consensus under the uniform averaging model is $n^{O(\frac{k}{\phi} \log d)} \log \frac{\tau(v_0)}{\epsilon}$.*

We introduce the idea of hitting diameter of a network sequence, which intuitively measures the number of time steps required for any person's opinion to have some influence over everyone else's.

Definition 6 (μ-Hitting Diameter). *Given a network sequence $\mathcal{G} = \{G_t : t \geq 0\}$, let P_t be the transition matrix associated with G_t under the uniform averaging model. Let $0 < \mu < \frac{1}{n}$. The μ-hitting diameter of the sequence, denoted by $\mathsf{HDiam}_\mu(\mathcal{G})$, is at most T, if for every $t \geq 0$, every entry of the product $P_{t+T-1} P_{t+T-2} \cdots P_t$ is at least μ.*

Lemma 2 (Hitting Diameter and Convergence Time). *Given a sequence $\mathcal{G}$ of networks with $\mathsf{HDiam}_\mu(\mathcal{G}) \leq T$, an initial configuration $v_0 \in \mathbb{R}^n$ and $\epsilon > 0$, the convergence time to achieve ϵ-consensus is $O(\frac{T}{n\mu} \log \frac{\tau(v_0)}{\epsilon})$.*

Proof. By Definition 6 and Fact 2, we have for all $t \geq 0$, $\tau_1(\prod_{j=t+T-1}^{t} P_j) \leq 1 - n\mu \leq \exp(-n\mu)$, where we have used the inequality $1 + x \leq e^x$ for all real x.

Therefore, by Fact 1 described in the preliminary $\tau_2(v_t) \leq \exp(-n\mu \lfloor \frac{t}{T} \rfloor) \cdot \tau(v_0)$, which is at most ϵ, when $t \geq \frac{T}{n\mu} \log \frac{\tau(v_0)}{\epsilon}$. □

Next, we show how to use this lemma to derive the convergence time for a specified class of networks.

Lemma 3 (Hitting Diameter for Union Vertex Expanders). *Suppose a network sequence $\mathcal{G} = \{G_t : t \geq 0\}$ has bounded degree d and k-union vertex expansion at least ϕ. Then, for $\mu = (\frac{1}{d+1})^{O(\frac{k \log n}{\phi})}$, $\mathsf{HDiam}_\mu(\mathcal{G}) = O(\frac{k \log n}{\phi})$.*

Proof. We show that for $T = O(\frac{k \log n}{\phi})$ and $\mu = (\frac{1}{d+1})^T$, $\mathsf{HDiam}_\mu(\mathcal{G}) \leq T$. Hence, it suffices to show that for any $t \geq T - 1$, every entry of the product $\mathcal{P}_t^T := P_t P_{t-1} \cdots P_{t-T+1}$ is at least μ.

For $1 \leq i \leq j$, we use the notation $P_{t,k}[i..j]$ to denote the product

$P_{t-(i-1)k}P_{t-(i-1)k-1}\cdots P_{t-jk+1}$. Observe that if T is a multiple of k, then $\mathcal{P}_t^T = P_{t,k}[1..\frac{T}{k}]$.

Observe that for each t, the transition matrix P_t obtained from G_t through the uniform averaging model has the following properties.

- For each i, $P_t[i,i] = \frac{1}{d_t[i]+1}$.
- For $i \neq j$, $P_t[i,j] > 0$ *iff* $\{i,j\} \in G_t$.
- Since G_t has bounded degree d, every non-zero entry of P_t is at least $\frac{1}{d+1}$.

Observe that we can view a matrix P as a directed graph $G(P)$, where $(u,v) \in G(P)$ *iff* $P[u,v] > 0$. Hence, $G(P_t)$ is a directed version of G_t with self-loop at every node added.

Given square matrices $A_1, A_2, \ldots, A_l$, observe that the (u,v)-th entry of the product $A_1A_2\ldots A_l$ is non-zero *iff* node v can be reached from node u in exactly l steps such that for every $1 \leq i \leq l$, only an edge from $G(A_i)$ can be used in step i.

Hence, it follows that for every i, $G(P_{t,k}[i])$ is a directed version of $\cup_{r=t-(i-1)k}^{t-ik+1} G_r$ with self-loops added, which from the hypothesis has vertex expansion at least ϕ.

Fix some nodes u and v. It follows from the vertex expansion property that there are at least $(1+\phi)$ nodes w such that the (u,w)-th entry of $P_{t,k}[1]$ is non-zero. Repeating this argument $T_0 := \lceil \log_{1+\phi} \frac{n}{2} \rceil$ times, it follows there are more than $\frac{n}{2}$ nodes w such that the (u,w)-th entry of $P_{t,k}[1..T_0]$ is non-zero.

By a reverse argument, it follows that there are more than $\frac{n}{2}$ nodes w such that the (w,v)-th entry of $P_{t,k}[T_0+1, 2T_0]$ is non-zero. Hence, by taking $T = 2T_0k = O(\frac{k\log n}{\phi})$, we have shown that the (u,v)-th entry of the product $\mathcal{P}_t^T = P_{t,k}[1..\frac{T}{k}]$ is non-zero. Observe that $\mathcal{P}_t^T$ is a product of T matrices, each of whose non-zero entry is at least $\frac{1}{d+1}$. Hence, we conclude that every entry of $\mathcal{P}_t^T$ is at least $\mu := (\frac{1}{d+1})^T$, as required. □

The connectivity of a network can also be measured by its eigenvalue gap, whose relationship with vertex expansion is given by the following lemma, which is a variation of the Cheeger's Inequality. We defer its proof to the full version.

Lemma 4 (Eigenvalue Gap Implies Vertex Expansion). *For a network G with bounded degree d, define the weight vector $w \in \mathbb{Z}^n$ by $w[i] = d[i]+1$ for each i. Let $P := I_n - W^{-1}L$, where $W = \mathsf{Diag}(w)$ and L is the Laplacian of G. Suppose there exists $0 < \eta < 1$, such that $|\lambda_1(P)| \leq 1-\eta$. Then, $\Phi(G) = \Omega(\frac{\eta}{d})$.*

Combining Lemma 4 and Theorem 2 gives the following corollary.

Corollary 3 (Convergence Time for Networks with Eigenvalue Gap). *Given a network sequence $\{G_t\}$ with bounded degree d, define the weight vector $w_t \in \mathbb{Z}^n$ by $w_t[i] = d_t[i]+1$ for each i. Let $P := I_n - W_t^{-1}L$, where $W_t = \mathsf{Diag}(w_t)$ and L_t is the Laplacian of G_t. Suppose there exists $0 < \eta < 1$, such that $|\lambda_1(P)| \leq 1-\eta$. Then, given an initial vector $v_0 \in \mathbb{R}^n$ and $\epsilon > 0$, the convergence time to achieve ϵ-consensus under the uniform averaging model is $n^{O(\frac{d}{\eta}\log d)} \log \frac{\tau(v_0)}{\epsilon}$.*

4.2 Random Networks

Our analysis for union networks can be easily extended for random networks. Specifically, we only require that the union vertex expansion property holds with some positive probability. Hence, our results also hold for random graphs with expansion property, such as $G_{n,p}$.

Definition 7 (Union Vertex Expansion with Probability ξ). *Given a sequence $\{G_t : t \geq 0\}$ of networks, and an integer $k \geq 1$, we say the sequence has k-union vertex expansion at least ϕ with probability $\xi > 0$, if for any $t \geq 0$, $\Phi(\cup_{r=t}^{t+k-1} G_r) \geq \phi$ holds with probability at least $\xi > 0$; moreover, the events involving different t's are independent, as long as the underlying G_r's involved are different.*

Theorem 3. *Suppose a network sequence $\{G_t\}$ has bounded degree d and there exist $\phi > 0$ and $\xi > 0$, such that the sequence has k-union vertex expansion at least ϕ with probability ξ. Then, given an initial vector $v_0 \in \mathbb{R}^n$ and $\epsilon > 0$, with all but negligible probability $\exp(-n^{\Theta(\frac{k \log d}{\phi \xi})})$, the convergence time to achieve ϵ-consensus is $n^{O(\frac{k \log d}{\phi \xi})} \log \frac{\tau(v_0)}{\epsilon}$.*

Proof. The theorem is obtained by proving probabilistic versions of Lemmas 3 and 2. We first argue that for $T = O(\frac{k \log n}{\xi \phi})$, for every $t \geq T-1$, with probability at least $\frac{1}{2}$, every entry of the product $\mathcal{P}_t^T := P_t P_{t-1} \cdots P_{t-T+1}$ is at least $\mu := (\frac{1}{d+1})^T$.

Consider a block of networks from the sequence of size k. By the hypothesis, the union graph over k networks has vertex expansion at least ϕ with probability at least ξ. From the proof of Lemma 3, we need $l := O(\frac{\log n}{\phi})$ such blocks in order to argue that every entry of the corresponding product of transition matrices is non-zero. Hence, by Chernoff Bound, if we have $\frac{2l}{\xi}$ such blocks, then with probability at least $1 - e^{-\Theta(l)} \geq \frac{1}{2}$, at least l blocks will have the expansion property. Since each block contains k networks, it follows that the $\frac{2l}{\xi}$ blocks contain $T = \frac{2l}{\xi} \cdot k = O(\frac{k \log n}{\xi \phi})$ networks, as required.

Let $P^{(T)}$ denote the product of a block of T transition matrices derived from the given sequence. We have just proved that with probability at least $\frac{1}{2}$, each entry in $P^{(T)}$ is at least μ. Hence, from Fact 2, we can conclude that with probability at least $\frac{1}{2}$, $\tau_1(P^{(T)}) \leq 1 - n\mu \leq e^{-n\mu}$.

Finally, if we are given an initial configuration vector v_0 and $\epsilon > 0$, the τ_2-measure of the configuration vector after multiplying by M such blocks of matrices (each block is a product coming from T transition matrices and each entry of the product is at least μ) is at most $\tau(v_0) \cdot e^{-Mn\mu}$, which is at most ϵ for $M = \Omega(\frac{1}{n\mu} \log \frac{\tau(v_0)}{\epsilon})$.

Hence, using Chernoff Bound again, with probability at least $1 - e^{-\Theta(M)} \geq 1 - \exp(-n^{\Theta(\frac{k \log d}{\phi \xi})})$, given $4M$ blocks of size T transition matrices, at least M such blocks will have a product such that every entry is at least μ.

Therefore, we conclude with all but negligible probability $\exp(-n^{\Theta(\frac{k \log d}{\phi \xi})})$, the convergence time to achieve ϵ-consensus is $O(MT) = n^{O(\frac{k \log d}{\phi \xi})} \log \frac{\tau(v_0)}{\epsilon}$. □

5 Experiments

In this section, we design experiments to simulate the behavior of our convergence models. In each simulation, we choose $\epsilon = 0.001$ and record the average convergence time to achieve ϵ-consensus. We observe in each case how the convergence time varies with n, the size of the network.

(1) $G_{n,p}$ under the Uniform Averaging Model. At time t, the network G_t is sampled independently from $G_{n,p}$, where $p = \frac{d}{n}$ $(d = 5, 10, 20)$. The result is in Figure 1.

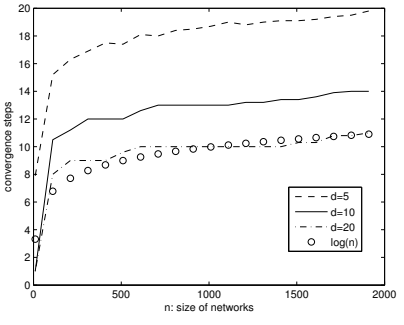


Fig. 1. $G_{n,p}$: $p = \frac{d}{n}$ under the Uniform Averaging Model

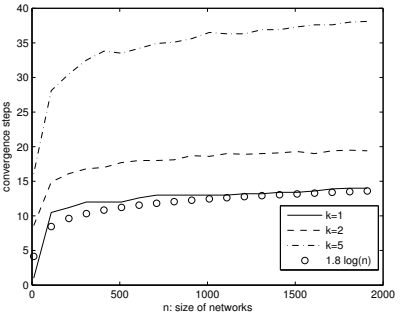


Fig. 2. k-Union $G_{n,p}$ (with $p = \frac{10}{n}$) under the Uniform Averaging Model

(2) k-Union $G_{n,p}$ under the Uniform Averaging Model. We fix $p = \frac{10}{n}$ and use the $G_{n,p}$ as the union network over k consecutive time steps. In particular, we sample a $G_{n,p}$ graph as before and divide the edge set (randomly) into k different sets $(k = 1, 2, 5)$, each of which forms the edge set of a network in one time step. The result is in Figure 2.

In all the experiments, we see that the convergence time grows logarithmically with the size of the networks. As a visual aid, the circles in each graph give a reference for logarithmic growth.

Acknowledgment. We would like to thank Kurt Mehlhorn for introducing the problem to us, and Konstantinos Panagiotou, Nikolaos Fountoulakis, Hing-Fung Ting and Tak-Wah Lam for useful discussion during the initial stages of this work.

References

1. Bertsekas, D., Tsitsiklis, J.N.: Parallel and Distributed Computation: Numerical Methods. Prentice Hall, Englewood Cliffs (1989)
2. Cao, M., Morse, A.S., Anderson, B.D.O.: Coordination of an asynchronous, multi-agent system via averaging. In: The Proceeding of 16th International Federation of Automatic Control World Congress, IFAC (2005)
3. Cao, M., Spielman, D.A., Morse, A.S.: A lower bound on convergence of a distributed network consensus algorithm. In: The Proceeding of the 44th IEEE Conference on Decision and Control, and European Control Conference, CDC-ECC, pp. 2356–2361 (December 2005)
4. Chazelle, B.: Natural algorithms. In: The Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, pp. 422–431 (2009)
5. Cortes, J.: Finite-time convergent gradient flows with applications to network consensus. Automatica 42, 1993–2000 (2006)
6. Cucker, F., Smale, S.: Emergent behavior in blocks. IEEE Transaction on Automatic Control 52, 852–862 (2007)
7. DeGroot, M.H.: Reaching a consensus. Journal of American Statistical Association 69(345), 118–121 (1974)
8. Fang, L., Antsaklis, P.: On communication requirements for multi-agent consensus seeking. In: The Workshop on Networked Embedded Sensing and Control, pp. 53–67 (2005)
9. Jadbabaie, A., Lin, J., Morse, A.S.: Coordination of groups of mobile autonomous agents using nearest neighbor rules. IEEE Transactions on Automatic Control 48(6), 988–1001 (2003)
10. Landau, H.J., Odlyzko, A.M.: Bounds for eigenvalues of certain stochastic matrices. Linear Algebra and its Applications 38, 5–15 (1981)
11. Liu, Y., Yang, Y.R.: Reputation propagation and agreement in mobile ad-hoc networks. Wireless Communications and Networking 3, 1510–1515 (2003)
12. Lorenz, D.A., Lorenz, J.: Convergence to consensus by general averaging. In: Bru, R., Romero-Vivo, S. (eds.) Positive Systems. LNCIS, vol. 389, pp. 91–99. Springer, Heidelberg (2009)
13. Moreau, L.: Stability of multiagent systems with time-dependent communication links. IEEE Transactions on Automatic Control 50, 169–182 (2005)
14. Nedić, A., Olshevsky, A., Ozdaglar, A., Tsitsiklis, J.: On distributed averaging algorithms and quantization effects. IEEE Transactions on Automatic Control 54, 2506–2517 (2009)
15. Olshevsky, A., Tsitsiklis, J.N.: Degree fluctuations and the convergence time of consensus algorithms (preprint)
16. Olshevsky, A., Tsitsiklis, J.N.: Convergence speed in distributed consensus and averaging. SIAM Journal on Control and Optimization 48(1), 33–55 (2009)
17. Reynolds, C.W.: Flocks, herds, and schools: A distributed behavioral model. Computer Graphics 21(4), 25–34 (1987)
18. Tanner, H.G., Jadbabaie, A., Pappas, G.J.: Flocking in fixed and switching networks. Electronic Communications in Probability 52, 863–868 (2006)
19. Vicsek, T., Czirok, A., Ben-Jacob, E., Cohen, I., Scochet, O.: Novel type of phase transition in a system of self-driven particles. Physical Review Letters 75(6), 1226–1229 (1995)

Linear Programming in the Semi-streaming Model with Application to the Maximum Matching Problem*

Kook Jin Ahn and Sudipto Guha

Department of Computer Information Sciences,
University of Pennsylvania, PA 19104
{kookjin,sudipto}@cis.upenn.edu

Abstract. In this paper, we study linear programming based approaches to the maximum matching problem in the semi-streaming model. The semi-streaming model has gained attention as a model for processing massive graphs as the importance of such graphs has increased. This is a model where edges are streamed-in in an adversarial order and we are allowed a space proportional to the number of vertices in a graph.

In recent years, there has been several new results in this semi-streaming model. However broad techniques such as linear programming have not been adapted to this model. We present several techniques to adapt and optimize linear programming based approaches in the semi-streaming model with an application to the maximum matching problem. As a consequence, we improve (almost) all previous results on this problem, and also prove new results on interesting variants.

1 Introduction

Analyzing massive data sets has been one of the key motivations for studying streaming algorithms. The streaming model is a computing model where we have random accessible memory sublinear in the input size and only sequential access to the input data is allowed. In recent years there has been significant interest in processing graph data, which arise in a number of scientific experimental setting, call records of telecoms etc., in the streaming model. In many emerging data analysis applications, large graphs are created based on implicit relationship between objects [5,20]. Subsequently, the goal is to find suitable combinatorial structure in this large implicit graph, e.g., maximum b-matchings were considered in [20]. The implicitly defined edges are often generated through "black box" transducers which have ill understood structure (or are based on domain specific information), but are prohibitive to store. A natural question in this regard is: Can we solve the combinatorial optimization problems without storing the edges explicitly? The reader would immediately observe the connection to "in place algorithms", which also poses the question of solving a problem

* Research supported in part by an NSF Award CCF-0644119, IIS-0713267 and a gift from Google.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 526–538, 2011.

using as small a space as possible excluding the input. In many massive data settings, or when the input is implicitly defined, we are faced with "in place algorithm with no random access or writes to the input". Multipass streaming algorithms seem well placed to answer these types of questions; concretely, can we solve maximum matching using small additional (to the input) space where we only make a few passes over the list of edges?

Graph problems were one of the early problems considered in the streaming model, and it was shown that even simple problems such as determining the connectedness of a graph requires $\Omega(n)$ space [18] (throughout this paper n will denote the number of vertices and m will denote the number of edges). This result holds even if a constant number of passes were allowed. The semi-streaming model [10,24] has emerged to be a model of choice in the context of graph processing – by allowing $\tilde{O}(n)$ space, which can be sublinear in the size of the input stream (of m edges) arriving in any (including adversarial) order. In recent years there has been several new results in this semi-streaming model, for example see [10,11,22,1,8]. Several of these papers address fundamental graph problems including matchings. These papers demonstrate a rich multidimensional tradeoff between the quality of the solution, the space required and the number of passes over the data (and of course, the running time). Yet, it is natural to ask: are there broad techniques that can be adapted to this model?

Our results: In this paper we answer both the questions posed above in the affirmative. We investigate primal–dual based algorithms for solving a subclass of linear programming problems on graphs. The maximum weighted matching (MWM) is a classic example of this – and although augmentation based techniques exist for matching problems, they become significantly difficult in the presence of weights (since we need to find shortest augmenting paths to avoid creating negative cycles) and the best previous result for this problem is a $\frac{1}{2} - \epsilon$ approximation using $O(\frac{1}{\epsilon^3})$ passes. The use of linear programming relaxation improves both the number of passes as well as the approximation factor, see Table 1. We also improve the number of passes for finding the maximum cardinality matching (MCM) in bipartite graphs by a significant amount.

Our Techniques: The matching problem has a rich literature, see [7,16,19,23], as well as fast, near linear time approximation algorithms [21,27,28,25,6]. However, these results use random access significantly and do not translate to results in the semi-streaming model, and newer ideas were used in [10,22,8] to achieve results in the semi-streaming model. We use the multiplicative-weights update meta-method surveyed in [4]. Over many years there has been a significant thrust in designing fast approximate schemes (have a $(1 + \epsilon)$ approximation for any given $\epsilon > 0$) for linear programming problems [26,29,13,17,12,4], to name a few. The meta-method uses the oracle to progressively improve a solution, but uses a (guessed) value of the optimal solution. The failure of the oracle either violates that guess or provides a proof of near-optimality. While the key intuition is to design a "streaming separation oracle", it is not clear how to implement (or define) such an oracle since there are number of conditions that need to be

Table 1. Summary of results: The required time is $\tilde{O}_\epsilon(m) = O(m \text{ poly}(\frac{1}{\epsilon}, \log n))$ for all results, unless otherwise noted. The space bounds of results presented elsewhere were not always obvious, and we have omitted reporting them. Note $n' = \min\{n, |OPT| \log \frac{1}{\epsilon}\}$ and $B = n$ for the uncapacitated case; otherwise $B = \sum_i b_i$. Please note that the result in [2] is subsequent to this paper and builds on the results herein.

Problem	Approx.	No. of Passes τ	space	paper	notes
Bipartite MCM (see all below)	$\frac{2}{3}(1-\epsilon)$	$O(\epsilon^{-1}\log\epsilon^{-1})$		[10]	
	$1-\epsilon$	$O(\epsilon^{-8})$		[8]	
	$1-\epsilon$	$O(\frac{1}{\epsilon^2}\log\log\frac{1}{\epsilon})$	$O\left(n'(\tau+\log n')\right)$	**here**	Sec. 5
MCM (see MWM below)	1/2	1			trivial
	$1-\epsilon$	$\left(\frac{1}{\epsilon}\right)^{1/\epsilon}$		[22]	
Bipartite MWM	$1-\epsilon$	$O(\epsilon^{-2}\log\epsilon^{-1})$	$O\left(n\left(\tau+\frac{\log n}{\epsilon}\right)\right)$	**here**	Sec. 5
Bipartite b-Matching	$1-\epsilon$	$O(\epsilon^{-3}\log n)$	$\tilde{O}\left(\frac{B}{\epsilon^3}\right)$	**here**	
MWM	1/4.91	1		[9]	
	$\frac{1}{2}(1-\epsilon)$	$O(\epsilon^{-3})$		[22]	
	$\frac{2}{3}(1-\epsilon)$	$O(\epsilon^{-2}\log\epsilon^{-1})$	$O\left(n\left(\tau+\frac{\log n}{\epsilon}\right)\right)$	**here**	from bipartite MWM
	$1-\epsilon$	$O(\epsilon^{-4}\log n)$	$O\left(n\left(\tau+\frac{\log n}{\epsilon}\right)\right)$	**here**	not $\tilde{O}_\epsilon(m)$ time
	$1-\epsilon$	$O(\epsilon^{-7}\log\epsilon^{-1}\log n)$	$O(\frac{n\tau}{\epsilon})$	[2]	
b-Matching	$1-\epsilon$	$O(\epsilon^{-4}\log n)$	$\tilde{O}\left(\frac{B}{\epsilon^4}\right)$	**here**	not $\tilde{O}_\epsilon(m)$ time

satisfied and they mandate random access and the overall scheme to obtain a good semi-streaming algorithm faces a number of roadblocks. First, the multiplicative update method typically requires super constant number of rounds (to prove feasibility) – this translates to superconstant number of passes. Reducing the number of passes to a constant requires that we recursively identify small and critical subgraphs. Second, for the weighted variants, it is non-trivial to simultaneously ensure that enough global progress is being made per pass, yet the computation in a pass is local and uses small space.

Roadmap: We revisit the multiplicative weights update method in Section 2. We then present a simple (but suboptimal in passes) application of this framework in Section 3. We reduce the number of passes by "simulating" multiple iterations of the multiplicative weights update method in a single pass in Section 4. We finally show how to remove the dependency on n in Section 5. We also present some extensions of the maximum matching problem in the full version of the paper [3]. Due to space constraints, all proofs are relegated to the full version as well.

2 The Multiplicative Weights Update Meta-method

In this section, we briefly explain the multiplicative weights update method; we follow the discussion presented by Arora, Hazan, and Kale [4]. Although the method can be applied to generalizations of linear programs (LPs), we restrict

our attention to LPs (primarily to reduce the number of parameters and nuances that need to be discussed). Suppose that we are given the following LP, its dual LP, and a guess of the optimal solution α, where $\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^n, \mathbf{c} \in \mathbb{R}^m$:

$$\text{LP:} \begin{cases} \min \mathbf{b}^{\mathrm{T}}\mathbf{x} \\ \text{s.t } \mathbf{A}^{\mathrm{T}}\mathbf{x} \geq \mathbf{c}, \quad \mathbf{x} \geq \mathbf{0} \end{cases} \qquad \text{Dual LP:} \begin{cases} \max \mathbf{c}^{\mathrm{T}}\mathbf{y} \\ \text{s.t } \mathbf{A}\mathbf{y} \leq \mathbf{b}, \quad \mathbf{y} \geq \mathbf{0} \end{cases}$$

The framework consists of multiple iterations of two-party game between the algorithm and an oracle. In each iteration, the algorithm provides weight u_i^t for each dual constraint (these weights correspond to a possibly infeasible primal solution). The oracle returns a dual feasible witness $\mathbf{y}^t$ which reduces the duality gap (see the definition of admissibility below). The failure to return a witness implies that the duality gap is small, or in other words the primal solution is (near) optimal. The framework is given in Algorithm 1. Note that the algorithm constructs a dual feasible solution $\frac{1}{T}\sum_t \mathbf{y}^t$ and proof of (near) optimality based on the proof of a small dual gap.

Algorithm 1. The Multiplicative Weights Update Meta-Method [4]

1: $u_i^1 = 1$ for all $i \in [n]$.
2: **for** $t = 1$ **to** T **do**
3: Given $\mathbf{u}^t$, the oracle returns an admissible dual witness $\mathbf{y}^t$. Note that $\mathbf{y}^t$ is not required to be feasible.
4: Let $\mathrm{M}(i, \mathbf{y}^t) = \mathbf{A}_i\mathbf{y}^t - b_i$ (for all i).
5: Assert $-\ell \leq \mathrm{M}(i, \mathbf{y}^t) \leq \rho$.
6: $u_i^{t+1} = \begin{cases} u_i^t(1+\epsilon)^{\mathrm{M}(i,\mathbf{y}^t)/\rho} & \text{if } \mathrm{M}(i, \mathbf{y}^t) \geq 0 \\ u_i^t(1-\epsilon)^{-\mathrm{M}(i,\mathbf{y}^t)/\rho} & \text{if } \mathrm{M}(i, \mathbf{y}^t) < 0 \end{cases}$
7: **end for**
8: Output $\frac{1}{T}\sum_t \mathbf{y}^t$ (along with any correction induced by failure of the oracle).

Definition 1. *We define* $\mathrm{M}(i, \mathbf{y}^t) = \mathbf{A}_i\mathbf{y}^t - b_i$ *to be the* **violation** *for dual constraint* i*. The* **expected violation** $\mathrm{M}(\mathcal{D}^t, \mathbf{y}^t)$ *is the expected value of* $\mathrm{M}(i, \mathbf{y}^t)$ *when choosing* i *with probability proportional to* u_i^t*, i.e.,* $\sum_i \frac{u_i^t}{\sum_j u_j^t}\mathrm{M}(i, \mathbf{y}^t)$*. The dual witness is defined to be* **admissible** *if it satisfies*

$$\mathrm{M}(\mathcal{D}^t, \mathbf{y}^t) \leq \delta, \quad \mathbf{c}^{\mathrm{T}}\mathbf{y}^t \geq \alpha, \quad \textit{and} \quad \mathrm{M}(i, \mathbf{y}^t) \in [-\ell, \rho] \textit{ with } \ell \leq \rho$$

ρ is defined as the width parameter of the oracle. The parameters ϵ and T depend on ρ, ℓ, and the desired error bound δ. Note that admissibility does not imply feasibility.

Theorem 1. *[4] Let $\delta > 0$ be an error parameter. Suppose that $\mathrm{M}(i, \mathbf{y}^t) \in [-\ell, \rho]$ with $\ell \leq \rho$. Then, after $T = \frac{2\rho \ln(n)}{\delta\epsilon}$ rounds and using $\epsilon = \min\{\frac{\delta}{4\ell}, \frac{1}{2}\}$, for any constraint i we have: $(1-\epsilon)\sum_t \mathrm{M}(i, \mathbf{y}^t) \leq \delta T + \sum_t \mathrm{M}(\mathcal{D}^t, \mathbf{y}^t)$.*

Since $\frac{1}{T}\sum_t \mathrm{M}(i, \mathbf{y}^t) = \mathrm{M}(i, \frac{1}{T}\sum_t \mathbf{y}^t)$ and $\mathrm{M}(\mathcal{D}^t\mathbf{y}^t) \leq \delta$ for all t, the above theorem implies that $\mathrm{M}(i, \frac{1}{T}\sum_t \mathbf{y}^t) \leq (1-\epsilon)^{-1}(2\delta) \leq 4\delta$ (dividing both the left

and right side of the inequality by T). This translates to the fact that the dual constraints have a small violation.

The $\ln n$ term arises from the slack in the initial weights – which is analyzed by a potential function $\Phi_i^t = u_i^t/(\sum_j u_j^t)$ and its upperbound Ψ_i; and the number of iterations depends on $\ln \frac{\Psi_i}{\Phi_i^1}$. The proof uses the following inequalities:

$$u_i^t \leq \sum_j u_j^t \leq \left(e^{\epsilon \sum_{t'<t} \mathrm{M}(\mathcal{D}^{t'}, \mathbf{y}^{t'})/\rho}\right) \sum_j u_j^1,\ u_i^1 = \frac{\sum_j u_j^1}{n},\ u_i^t \approx u_i^1 e^{\epsilon \sum_{t'<t} \mathrm{M}(i, \mathbf{y}^{t'})/\rho}$$

Eliminating the dependence on n requires modifying the potential function (or the starting point or both), which we do in Section 5. The oracle is the problem-specific part of the framework and the overall efficiency of the algorithm depends on the oracle. Moreover, note that the multiplicative weights update method produces an explicit dual solution (and only verifies primal feasibility).

3 Warming Up: $O(\frac{1}{\epsilon^3} \log n)$-Pass Algorithms

In this section, we provide an $(1-\epsilon)$-approximation algorithm for (bipartite) MCM and MWM that runs in $O(\frac{1}{\epsilon^3} \log n)$ passes. Recall that the multiplicative update problem solves the dual. We formulate the primal LP (LP1 and LP3) to be the dual of the actual LP for MCM and MWM(LP2 and LP4). The integrality gap of LP2(and LP4) is one. We first present an algorithm for MCM and then generalize the algorithm for MWM.

$$\begin{aligned} &\min \textstyle\sum_i x_i \\ &\text{s.t } x_i + x_j \geq 1\ \forall (i,j) \in E \\ &\quad x_i \geq 0 \end{aligned} \qquad \text{(LP1)} \qquad \begin{aligned} &\max \textstyle\sum_{(i,j)\in E} y_{ij} \\ &\text{s.t } \textstyle\sum_{j:(i,j)\in E} y_{ij} \leq 1\ \forall i \\ &\quad y_{ij} \geq 0 \end{aligned} \qquad \text{(LP2)}$$

$$\begin{aligned} &\min \textstyle\sum_i x_i \\ &\text{s.t } x_i + x_j \geq w_{ij}\ \forall (i,j) \in E \\ &\quad x_i \geq 0 \end{aligned} \qquad \text{(LP3)} \qquad \begin{aligned} &\max \textstyle\sum_{(i,j)\in E} w_{ij} y_{ij} \\ &\text{s.t } \textstyle\sum_{j:(i,j)\in E} y_{ij} \leq 1\ \forall i \\ &\quad y_{ij} \geq 0 \end{aligned} \qquad \text{(LP4)}$$

The simple case of MCM: We apply the multiplicative weights method [4] with the oracle provided in Algorithm 2. Recall that the algorithm returns a (approximately) feasible solution for LP2 after T rounds. We can compute a maximal matching in one pass in the semi-streaming model in $O(m)$ time.

The intuition behind the oracle will be used for all the algorithms. Since we are trying to find a solution for LP2, the oracle must choose a subset S of edges. The critical properties and actions that need to be determined then are (i) admissibility and (ii) verification of near optimality (or that the guess of α is wrong). If S satisfies the following conditions, we achieve those goals.
(Admissibility): Each node i is adjacent to at most one edge in S (defines ℓ, ρ).
(Verification): For each edge (i,j), we pick at least one edge adjacent to it.

Any **maximal** matching in $E_{violate}$ satisfies both conditions. Since we consider the violated edges only, we improve the dual gap and the algorithm is natural.

Algorithm 2. Oracle for LP1

1: Let $\mathbf{x}_i = \frac{\alpha}{\sum_j u_j^t} u_i^t$. Let $E_{violated} = \{(i,j) | \mathbf{x}_i + \mathbf{x}_j < 1\}$.
2: Find a **maximal matching** S in $E_{violated}$. Let $\Delta = |S|$.
3: **if** $\Delta < \delta\alpha$ **then**
4: The oracle reports failure. We indicate the verification step: For each $(i,j) \in S$, increase $\mathbf{x}_i$ and $\mathbf{x}_j$ by 1 and return $\{\mathbf{x}_i\}$ as the overall primal solution.
5: **else**
6: Return $\mathbf{y}_{ij} = \alpha/\Delta$ for $(i,j) \in S$ and $\mathbf{y}_{ij} = 0$ otherwise.
7: **end if**

Lemma 1. *The oracle for LP1 returns an admissible solution with $\ell = 1$ and $\rho = 1/\delta$.*

Lemma 2. *If $\Delta < \delta\alpha$, a feasible solution for LP1 with value at most $(1+2\delta)\alpha$ is returned.*

Theorem 2. *In $T = O(\frac{1}{\epsilon^3} \log n)$ passes, we find either a feasible solution for LP1 with value at most $(1+\epsilon)\alpha$ or a feasible solution for LP2 with value at least $(1-\epsilon)\alpha$. This uses $O(nT)$ space and $O(mT)$ time. This gives us a $T+1$ pass $O(\frac{S}{\epsilon})$ space algorithm for $(1+\epsilon)$ approximation to maximum cardinality matching in bipartite graphs (computing only the size requires less space).*

The not so simple case of MWM: We generalize Algorithm 2 to construct an oracle for LP3. In the weighted graph, the verification condition must be strengthened to handle the complication introduced by the edge weight. In LP2, if we increase x_i by 1, all the edges adjacent to i are satisfied. It is not true in LP4. However, trying to fix the verification condition by itself does not help, any change also has to ensure the admissibility condition. For a set of edges S, let $w(S) = \sum_{(i,j) \in S} w_{ij}$ denote the total weight for that set.

(Weighted Admissibility of S): There exists a matching S' contained in S such that $w(S') = \Omega(w(S))$. We use S' to update the primal variables.

(Weighted Verification of S): For each violated edge (i,j), we pick at least one edge adjacent to it whose weight is $\Omega(w_{ij})$. We need all of S in this case.

The above two conditions are the conditions modified for LP4. If the oracle satisfies the above two conditions, we can use S for the verification step and use S' for constructing a dual witness. In order to satisfy the modified conditions, we partition edges depending on their weights.

Definition 2. *An edge (i,j) is in* **tier** *k if $\alpha/2^k < w_{ij} \leq \alpha/2^{k-1}$.*

We ignore edges with tier greater than $\log \frac{n}{\delta}$. The ignored edges contribute at most δ fraction of the optimal solution. Algorithm 3 is the oracle for LP3.

Lemma 3. *For every violated edge (i,j), we pick at least one edge adjacent to it whose weight is $\Omega(w_{ij})$.*

Algorithm 3. Oracle for LP3

Let $x_i = \frac{\alpha}{\sum_j u_j^t} u_i^t$.
Let $E_{violated,k} = \{(i,j) | x_i + x_j < w_{ij}, \alpha/2^k < w_{ij} \leq \alpha/2^{k-1}\}$.
Find a maximal matching S_k in $E_{violated,k}$ for each $k = 1, \cdots, \lceil \log \frac{n}{\delta} \rceil = O(\log n)$.
Let $S = \cup_k S_k, \Delta = w(S)$.
if $\Delta < \delta\alpha$ **then**
  For each $(i,j) \in S$, increase x_i and x_j by $2w_{ij}$. Return $\{x_i\}$.
else
  repeat
    Pick a heaviest edge (i,j) from S and add it to S'
    Eliminate all edges adjacent to i or j from S.
  until $S = \emptyset$
  Return $y_{ij} = \alpha/w(S')$ for $(i,j) \in S'$ and $y_{ij} = 0$ otherwise.
end if

Lemma 4. *There exists a matching $S' \subseteq S$ such that $w(S') = \Omega(w(S))$.*

The rest of the argument is almost identical to MCM and proof of Theorem 2 (ρ increases by a constant factor) with the exception of the final rounding scheme for we use the recent result of [6]. The space bound increases due to running $O(\frac{1}{\epsilon})$ copies of the oracle, each of which uses $O(n \log n)$ space. Note that our algorithm did not use bipartiteness (except in the proof of the integrality gap of the MWM formulation).

Theorem 3. *In $T = O(\frac{1}{\epsilon^3} \log n)$ passes, and $O(\frac{nT}{\epsilon} + \frac{n}{\epsilon} \log n)$ space we can compute a $(1+\epsilon)$ approximation for maximum weighted matching in bipartite graphs (again the size can be computed in less space). This implies a $\frac{2}{3}(1-\epsilon)$ result for general graphs using the integrality gap results of [14,15].*

4 Reducing Passes and Space Requirement

In the previous section, the algorithm executes the oracle in one pass and consequently, the number of passes is equal to the number of iterations (except the first pass to guess α). In this section, we first reduce the number of passes by executing multiple iterations of the algorithm in one pass – this can be viewed as making a "step" which is significantly larger than what is provided by the basic analysis in the last section. Subsequently we address the space required.

Consider the two conditions for the oracle given in the previous section, and for the sake of exposition, only consider the cardinality case. Suppose that we just performed an update based on a dual witness $\mathbf{y}$. The admissibility condition remains satisfied as long as the edges (i,j) in $\mathbf{y}$ satisfy $\mathbf{x}_i + \mathbf{x}_j < 1$. The verification condition is not used as long as we have such a matching. In other words, we can use the same matching until one of its edge becomes tight, that is, satisfies the corresponding constraint $\mathbf{x}_i + \mathbf{x}_j \geq 1$. One caveat of this argument is that the "progress" (say, measured in the number of iterations of the meta

method that can be simulated in a single pass) could be small when $\mathbf{x}_i + \mathbf{x}_j$ is close to 1. We have to adjust $\mathbf{y}_{ij}$ in order to guarantee a minimum progress per pass. Observe that this idea automatically brings up the notion of weights. The high-level idea for the oracle is similar to the construction in Section 3 – but there are significant differences and two major issues arise.

- First, we cannot use the uniform growth of the entries $\mathbf{y}$ as in Section 3. Suppose that S contains (i,j) and (i',j') where $1-\mathbf{x}_{i'}-\mathbf{x}_{j'}$ is greater than $1-\mathbf{x}_i-\mathbf{x}_j$. If we assign large values to $\mathbf{y}_{ij}$ and $\mathbf{y}_{i'j'}$, it decreases the number of iterations per pass (due to normalization). If we assign small values to $\mathbf{y}_{ij}$ and $\mathbf{y}_{i'j'}$, it increases the total number of iterations and it may result in inadmissible $\mathbf{y}$, i.e., $\mathbf{c}^T\mathbf{y} < \alpha$.
- Second, we have to modify the verification condition in Section 3 so that the condition handles the values of $w_{ij} - \mathbf{x}_i - \mathbf{x}_j$ and keep the increase of the solution minimal. For example, (again using the cardinality case as an example) increasing the value of $\mathbf{x}_i$ and $\mathbf{x}_j$ less than 1 in the verification step can result in an infeasible primal solution. On the other hand, increasing $\mathbf{x}_i$ and $\mathbf{x}_j$ by 1 can result in a larger approximation factor.

In what follows, we avoid both the issues using "layers" of violations.

Definition 3. *We define* $\tilde{v}_{ij} = (w_{ij} - x_i - x_j)/w_{ij}$ *to be a* **fractional violation** *and* $v_{ij} = \tilde{v}_{ij} w_{ij} = w_{ij} - x_i - x_j$ *be a* **(weighted) violation** *of an edge* (i,j). *An edge* (i,j) *is in* **tier** k *if* $\alpha/2^k < v_{ij} \leq \alpha/2^{k-1}$. *For any set* S, *we define* $V(S) = \sum_{(i,j)\in S} v_{ij}$. *The fractional violation will be used in increasing* $\mathbf{y}$ *and the weighted violation will be used in the verification step.*

We now use **relaxed** conditions of (weighted) Admissibility and Verification of a witness S as in Section 3, i.e., we choose a set of edges S such that (i) there exists a matching $S' \subseteq S$ with $V(S') = \Omega(V(S))$ and (ii) for each violated edge (i,j), we pick at least one edge adjacent to it in S whose violation weight is $\Omega(v_{ij})$. We can now prove the following:

Theorem 4. *Theorem 3 holds with* $T = O(\frac{1}{\epsilon^2}\log n)$.

4.1 Reducing Space Required to Run Parallel Guesses

The basic intuition is to treat the admissibility and verification conditions differently. Note that the verification condition is applied only at the terminal step of an oracle. In what follows we show how to preserve the admissibility condition across different values of the guessed parameter α, and run the $O(\frac{1}{\epsilon})$ guesses (in Theorem 4) in parallel. We begin with the following definition:

Definition 4. *A sequence* $\mathbf{y}^1, \mathbf{y}^2, \cdots, \mathbf{y}^t$ *is* **admissible** *if all* $\mathbf{y}$ *are admissible when we apply* $\mathbf{y}^1, \mathbf{y}^2, \cdots, \mathbf{y}^t$ *in the given order.*

Lemma 5. *Let* α, α' *be guesses of the optimal solution with* $\alpha > \alpha'$. *If a sequence* $\mathbf{y}^1, \mathbf{y}^2, \cdots, \mathbf{y}^t$ *is admissible for* α, *the sequence is also admissible for* α'.

Algorithm 4. Improved Oracle for LP3(and LP1)

Let $x_i = \frac{\alpha}{\sum_j u_j^t} u_i^t$.
Let $E_{violated,k} = \{(i,j) | (i,j) \text{ is in tier } k\}$. for $k = 1, \cdots, K = \lfloor \log_2 \frac{n\alpha}{\delta} \rfloor$
Find a **maximal matching** S_k in each $E_{violated,k}$.
Let $S = \cup_k S_k$and $\Delta = V(S)$.
if $\Delta < \delta\alpha$ **then**
    The oracle reports failure. We indicate the verification step: For each $(i,j) \in S$, increase x_i and x_j by $2v_{ij}$. Increase all x_i by $\frac{\delta\alpha}{n}$ and return $\{x_i\}$ as the overall primal solution.
else
    $S' \leftarrow \emptyset$.
    for $k = 1$ **to** K **do**
        $S' \leftarrow S' \cup S_k$.
        For $k' > k$, eliminate all edges in S'_k that are adjacent to an edge in S_k.
    end for
    Let $\Delta' = V(S')$.
    Return $y_{ij} = \tilde{v}_{ij}\alpha/\Delta'$ for $(i,j) \in S'$ and $y_{ij} = 0$ otherwise.
end if

Algorithm 5. Overall Algorithm for MWM. Note that the update rule corresponds to $1/\delta$ iterations of the original method (Algorithm 1) and the witness (and the violations) are not updated in between.

$u_i^1 = 1$ for all $i \in [n]$.
for $t = 1$ **to** δT **do**
    Given u_i^t, the oracle (for LP3) returns a dual witness $\mathbf{y}^t$.
    Let $\mathrm{M}(i, \mathbf{y}^t) = \mathbf{A}_i \mathbf{y}^t - b_i$.
    Assert $-1 \leq \mathrm{M}(i, \mathbf{y}^t) \leq 5/\delta$.
    $u_i^{t+1} = \begin{cases} u_i^t (1+\epsilon)^{\mathrm{M}(i,\mathbf{y}^t)/5} & \text{if } \mathrm{M}(i,\mathbf{y}^t) \geq 0 \\ u_i^t (1-\epsilon)^{-\mathrm{M}(i,\mathbf{y}^t)/5} & \text{if } \mathrm{M}(i,\mathbf{y}^t) < 0 \end{cases}$
end for
Output $\frac{1}{T}\sum_t \mathbf{y}^t$ (along with any correction induced by failure of the oracle).

The algorithm: We start with α being the upperbound of the maximum matching. Each time the oracle fails, we reduce α by $(1-\delta)$ factor while keeping the weights of constraints and $\{\mathbf{y}^t\}$ fixed. This is possible since the sequence of $\mathbf{y}$ remains admissible with the same width parameter. The total number of successful iterations remains the same but we need additional iterations for oracle failures which is at most $O(\frac{1}{\epsilon})$.

Theorem 5. *Theorem 4 holds with space* $O(n(T + \log n))$.

5 Removing the Dependency on n

In this section, we present an algorithm for MCM where the number of passes does not depend on the number of nodes. We use the same algorithm as in

Section 4 but we use a subgraph of the input graph and apply a different analysis. Recall the analysis of the multiplicative weight update method and the discussion regarding the proof of Theorem 1. The number of iterations is proportional to $\ln \frac{\Psi_i}{\Phi_i^1}$ where $\Phi_i^t = (\sum_j u_j^t)/u_i^t$ and Ψ_i is the upperbound of Φ_i^t. By the definition of u_i^1, $\Phi_i^1 = 1/n$. x_i increases upto $1 + O(\delta)$ while the sum of x_i is maintained as α. Hence $\Psi_i = O(1/\alpha)$.

The simpler case of cardinality matching: The above implies that if we can identify a small subgraph (in the number of vertices) then we can remove the dependency on n. Indeed, this very simple approach works for maximum cardinality matching. Consider the Algorithm 6 and the following lemma:

Algorithm 6. Constant-pass Algorithm for MCM

1: Find a maximal matching and find a 2 approximation of OPT.
2: Let S_0 be the set of vertices that are matched.
3: **for** $t = 1$ **to** $O(\log \frac{1}{\delta})$ **do**
4: Find a maximal matching between S_{t-1} and $V - S_{t-1}$. Let T_t be the set of vertices in the maximal matching.
5: $S_t = S_{t-1} \cup T_t$.
6: **end for**
7: Let G' be a subgraph induced by S_T. This can be achieved by filtering the stream.
8: Run Algorithm 5 on G' (with modification of Section 4.1).

Lemma 6. *Let OPT_S denote the maximum matching in the subgraph induced by the vertex set $S \subseteq V$, then (using the notation of Algorithm 6), we have $OPT_V - OPT_{S_{t+2}} \leq \frac{2}{3}(OPT - OPT_{S_t})$. (This proof does not use bipartiteness.)*

If Lemma 6 is repeated $O(\log \frac{1}{\delta})$ times (as in Algorithm 6), the difference between the optimal solution in G and the optimal solution in the subgraph G' is at most δOPT. The size of each maximal matching is $O(\alpha)$ and we repeat $O(\log \frac{1}{\delta})$, the subgraph contains at most $O(\alpha \log \frac{1}{\delta})$ vertices. The number of passes to find the subgraph is $O(\log \frac{1}{\delta})$ since we can find a maximal matching in one pass. Using $\delta = O(\epsilon)$, and Theorem 5, we obtain the following theorem:

Theorem 6. *In $T = O(\frac{1}{\epsilon^2} \log \log \frac{1}{\epsilon})$ passes, and $O(n'(T + \log n'))$ space we can compute a $(1 + \epsilon)$ approximation for the maximum cardinality matching in bipartite graphs where $n' = \min\{n, |OPT| \log \frac{1}{\epsilon}\}$.*

The case of Maximum Weighted Matching. This case is significantly more difficult and we cannot in general find a small subgraph. The subgraph will now be expressed implicitly using the vertex weights u_i as proxy. Intuitively, instead of staring from an uniform random sample of the constraints, we will start from a weighted sample. Before proceeding further, for the rest of this section we assume that the weights are discrete, i.e., $w_{ij} \in \{1, (1 + \delta), \cdots, (1 + \delta)^L\}$ where $L = O(\frac{1}{\delta} \log n)$ to simplify the analysis. The discretization of the weights reduces

the optimal solution by at most $(1-\delta)$ factor. This can be achieved with one extra pass and a (constant factor) estimate of the maximum weighted matching (denoted henceforth as $\mathcal{M}$), and then we ignore the edge weights which are smaller than δ/n times $w(\mathcal{M})$. We ensure that:

C1: $\sum_i u_i^1 = \gamma w(\mathcal{M})$.
C2: Let $G' = (V, E')$ be a subgraph that consists of edges (i,j) such that $w_{ij} \leq u_i^1, u_j^1$. Then, G' contains a matching with weight at least $(1-\delta)w(\mathcal{M})$.

By the definition of u_i^1, $\Phi_i^1 = \Omega(u_i^1/\gamma\alpha)$. Since $w_{ij} \leq u_i^1$ for all $(i,j) \in E$, x_i increases upto $O(u_i^1)$ while the sum of x_i is maintained as α. So $\Psi_i = O(u_i^1/\alpha)$. Hence, we obtain an $(1-\epsilon)$-approximation in $O(\frac{1}{\epsilon^2}\log\gamma)$ passes. The Algorithm 7 that computes the weights of the vertices is similar to Algorithm 6, but is significantly non-trivial. The next two lemmas are the crux of the argument.

Algorithm 7. Finding a subgraph G'. q is a parameter which will be defined later.

1: (i,j) is in **level** k if $w_{ij} = (1+\delta)^k$
2: **for each level** $k = 0, 1, \cdots, L$ **in parallel do**
3: Find a maximal matching.
4: Let C_k be the set of nodes matched in the maximal matching.
5: Let $S_k^1 = C_k$
6: **for** $t = 1$ **to** q **do**
7: Find a maximal matching between C_k and $V - S_k^t$.
8: Let T_k^t be the set of nodes matched in the maximal matching.
9: $S_k^{t+1} = S_k^t \cup T_k^t$.
10: **end for**
11: **end for**
12: Let $u_i^1 = (1+\delta)^k$ for the maximum k with $i \in S_k^q$
13: Let $G' = (V, E')$ where $E' = \{(i,j) : w_{ij} \leq u_i^1, u_j^1\}$.
14: Run Algorithm 5 on G' with initial weights u_i^1 and return its result.

Lemma 7. $\sum_i u_i^1 = O(\frac{q}{\delta})w(\mathcal{M})$. *(Bipartiteness is not used in this proof.)*

Lemma 8. *G' contains a matching with weight at least* $\left(1 - \frac{O(\delta^{-1}\log\delta^{-1})}{q}\right) w(\mathcal{M})$.

With $q = O(\frac{1}{\delta^2}\log\frac{1}{\delta})$ and $\gamma = O(\frac{1}{\delta^3}\log\frac{1}{\delta})$, we satisfy both $C1$ and $C2$ and obtain an $O(\frac{1}{\delta^2}\log\frac{1}{\delta})$-pass algorithm for testing the linear program MWM. We set $\delta = O(\epsilon)$ and note, it takes q passes and $O(nL) = O(\frac{n\log n}{\epsilon})$ space to find the subgraph G'. Using Theorem 5 and summarizing,

Theorem 7. *In $T = O(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})$ passes, and $O(n(T + \frac{\log n}{\epsilon}))$ space we can compute a $(1+\epsilon)$ approximation for the maximum weighted matching in bipartite graphs. This translates to a $\frac{2}{3}(1-\epsilon)$ approximation for general graphs.*

References

1. Ahn, K.J., Guha, S.: Graph sparsification in the semi-streaming model. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 328–338. Springer, Heidelberg (2009)
2. Ahn, K.J., Guha, S.: Laminar families and metric embeddings: Non-bipartite maximum matching problem in the semi-streaming model (manuscript, 2011), http://arxiv.org/abs/1104.4058
3. Ahn, K.J., Guha, S.: Linear programming in the semi-streaming model with application to the maximum matching problem. To appear in the Proceedings of ICALP (2011), http://arxiv.org/abs/1104.2315
4. Arora, S., Hazan, E., Kale, S.: The multiplicative weights update method: a meta algorithm and applications (2005), http://www.cs.princeton.edu/~arora/pubs/MWsurvey.pdf
5. Belkin, M., Niyogi, P.: Towards a theoretical foundation for laplacian based manifold methods. J. Comput. System Sci., 1289–1308 (2008)
6. Duan, R., Pettie, S.: Approximating maximum weight matching in near-linear time. In: FOCS, pp. 673–682 (2010)
7. Edmonds, J.: Maximum matching and a polyhedron with 0,1-vertices. Journal of Research of the National Bureau of Standards 69, 125–130 (1965)
8. Eggert, S., Kliemann, L., Srivastav, A.: Bipartite graph matchings in the semi-streaming model. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 492–503. Springer, Heidelberg (2009)
9. Epstein, L., Levin, A., Mestre, J., Segev, D.: Improved approximation guarantees for weighted matching in the semi-streaming model. In: Proc. of STACS, pp. 347–358 (2010)
10. Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: On graph problems in a semi-streaming model. Theor. Comput. Sci. 348(2-3), 207–216 (2005)
11. Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: Graph distances in the data-stream model. SIAM J. Comput. 38(5), 1709–1727 (2008)
12. Fleischer, L.K.: Approximating fractional multicommodity flow independent of the number of commodities. SIAM J. Discret. Math. 13(4), 505–520 (2000)
13. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci. (JCSS) 55(1), 119–139 (1997)
14. Füredi, Z.: Maximum degree and fractional matchings in uniform hypergraphs. Combinatorica 1(2), 155–162 (1981)
15. Füredi, Z., Kahn, J., Seymour, P.D.: On the fractional matching polytope of a hypergraph. Combinatorica 13(2), 167–180 (1993)
16. Gabow, H.N.: Data structures for weighted matching and nearest common ancestors with linking. In: SODA, pp. 434–443 (1990)
17. Garg, N., Könemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In: Proc. FOCS, pp. 300–309 (1998)
18. Henzinger, M., Raghavan, P., Rajagopalan, S.: Computing on data streams (1998)
19. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. 2(4), 225–231 (1973)
20. Jebara, T., Wang, J., Chang, S.-F.: Graph construction and b-matching for semi-supervised learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, pp. 441–448 (2009)
21. Kalantari, B., Shokoufandeh, A.: Approximation schemes for maximum cardinality matching. Technical Report LCSR-TR-248, Laboratory for Computer Science Research, Department of Computer Science. Rutgers University (1995)

22. McGregor, A.: Finding graph matchings in data streams. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX 2005 and RANDOM 2005. LNCS, vol. 3624, pp. 170–181. Springer, Heidelberg (2005)
23. Micali, S., Vazirani, V.V.: An $O(\sqrt{|V||E|})$ algorithm for finding maximum matching in general graphs. In: FOCS, pp. 17–27 (1980)
24. Muthukrishnan, S.: Data streams: Algorithms and applications. Foundations and Trends in Theoretical Computer Science 1(2) (2005)
25. Pettie, S., Sanders, P.: A simpler linear time 2/3-epsilon approximation for maximum weight matching. Inf. Process. Lett. 91(6), 271–276 (2004)
26. Plotkin, S.A., Shmoys, D.B., Tardos, É.: Fast approximation algorithms for fractional packing and covering problems. In: FOCS, pp. 495–504 (1991)
27. Preis, R.: Linear time 1/2 -approximation algorithm for maximum weighted matching in general graphs. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 259–269. Springer, Heidelberg (1999)
28. Vinkemeier, D.E.D., Hougardy, S.: A linear-time approximation algorithm for weighted matchings in graphs. ACM Transactions on Algorithms 1(1), 107–122 (2005)
29. Young, N.E.: Randomized rounding without solving the linear program. In: Proc. SODA, pp. 170–178 (1995)

Restoring Pure Equilibria to Weighted Congestion Games

Konstantinos Kollias and Tim Roughgarden

Stanford University, Stanford CA 94305, USA
kkollias@stanford.edu, tim@cs.stanford.edu

Abstract. Congestion games model several interesting applications, including routing and network formation games, and also possess attractive theoretical properties, including the existence of and convergence of natural dynamics to a pure Nash equilibrium. Weighted variants of congestion games that rely on sharing costs proportional to players' weights do not generally have pure-strategy Nash equilibria. We propose a new way of assigning costs to players with weights in congestion games that recovers the important properties of the unweighted model. This method is derived from the Shapley value, and it always induces a game with a (weighted) potential function. For the special cases of weighted network cost-sharing and atomic selfish routing games (with Shapley value-based cost shares), we prove tight bounds on the price of stability and price of anarchy, respectively.

Keywords: congestion games, network design, Shapley value.

1 Introduction

Congestion games are a well-studied generalization of several game-theoretic models, including some fundamental network formation games and routing games. In the standard model [19], there is a ground set of resources, and each player has a set of allowable strategies, each of which is a subset of resources. For example, the strategies of a player could correspond to the paths of a network with a given source and sink. Each resource has a per-user cost that depends on the number of players that use it, and the goal of each player is to minimize the sum of the resources' costs in its strategy, given the strategies chosen by the other players. In atomic selfish routing games [20,22], strategies correspond to paths and resource cost functions $c_e(\cdot)$ are nondecreasing. In network cost-sharing games [2], strategies correspond to paths and the (decreasing) cost functions have the form $c_e(f_e) = \gamma_e/f_e$, where γ_e is the fixed installation cost of an edge e and f_e is the number of players that share it.

A *pure Nash equilibrium (PNE)* is a strategy profile such that no player can decrease its cost via a unilateral deviation. Many games, such as "Rock-Paper-Scissors", have no PNE. Rosenthal [19] used a potential function argument to show that every congestion game — and thus every atomic selfish routing and

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 539–551, 2011.

network cost-sharing game — has at least one PNE. Moreover, several natural dynamics are guaranteed to converge to a PNE in congestion games [14,15].

Every player of a congestion game imposes the same load on a resource. There are many motivations for relaxing this assumption and allowing non-uniform resource consumption: for example, in a network context, players could have different durations of resource usage, different bandwidth requirements, or different contracts with the network provider. Almost all research to date has modeled non-uniform players in congestion-type games through *proportional cost sharing* [1,2,3,5,8,10,13,15]. The first assumption in proportional cost sharing is that each player i has a positive weight w_i, with larger weights indicating larger resource usage. To explain the second assumption in a general way, let $C_e(S_e)$ denote the joint cost incurred by the subset S_e of users of the resource e. For example, in a network cost-sharing game, $C_e(S_e)$ is the fixed cost γ_e provided S_e is non-empty (and is 0 otherwise). In (weighted) atomic selfish routing, $C_e(S_e)$ is $f_e \cdot c_e(f_e)$, where $c_e(\cdot)$ is the per-flow unit resource cost function and $f_e = \sum_{i \in S_e} w_i$ is the total weight of the players using e. Proportional cost sharing dictates that each player $i \in S_e$ pays a $w_i / \sum_{j \in S_e} w_j$ fraction of $C_e(S_e)$ for the resource e.

Unfortunately, most of the attractive theoretical properties of congestion games do not carry over to their weighted counterparts with proportional cost sharing. Network cost-sharing games with at least three players need not have a PNE [5]. Even when PNE do exist in such games, they can be much costlier (relative to an optimal solution) than in the unweighted case [2,5]. Atomic selfish routing games with weighted players do not generally have PNE [9,10,20], except when all cost functions are affine [7] and in some other isolated special cases [10].

Guaranteed existence of PNE is an important property. There are, of course, more general equilibrium concepts like the mixed-strategy Nash equilibrium that are guaranteed to exist in every finite game, but randomized solution concepts suffer from well-known drawbacks in interpretation and implementation (see e.g. [18, §3.2]). Particularly when designing or influencing the game being played — the standard interpretation of network cost-sharing games [2,5,6] and also a well-motivated one for routing games [16,17,24] — there is good reason to make design decisions (e.g., queueing policies [24]) that guarantee the existence of and convergence of natural dynamics to a PNE.

We propose a new way of assigning costs to players with weights in congestion-type games, which is derived from the Shapley value. We call the resulting class of games *SV weighted congestion games*. Extending work of Hart and Mas-Colell [11], we show that every SV weighted congestion game admits a (weighted) potential function. The existence of and convergence of natural dynamics to a PNE in every such game follow immediately.

For example, for the special case of atomic selfish routing games, we derive the cost shares for the users S_e of edge e by applying the standard Shapley value (defined in the next section) to the cost function C_e above with the player set S_e. Since the incremental effect of a player on the joint cost is increasing in

its weight, so is its cost share. These Shapley value-based cost shares coincide with proportional shares when all per-user cost functions are affine, but not otherwise. These observations explain the previously mysterious fact that the traditional proportional cost shares always yield a potential game if and only if all cost functions are affine [7,10].

For the special case of network cost-sharing games, the joint cost function C_e is insensitive to players' weights. To introduce weight-dependent cost shares, we use the *weighted* Shapley value [12,23], which averages over orderings of the players in a non-uniform way (see the next section for a definition). The resulting cost shares are increasing in weight, and coincide with proportional shares (for all weight vectors) if and only if there are at most two players. Again, these facts demystify the previously observed phase transition between the cases of two players [2] and three or more players [5] for proportional cost shares.

We also provide tight bounds on the inefficiency of equilibria in SV weighted network cost-sharing and atomic selfish routing games. For network cost-sharing games our results are tight in a strong sense: for every number k of players and *every* weight vector w, we characterize the worst-case price of stability (POS) — the ratio between the cost of the best PNE and of an optimal solution — in games with player weight vector w. For each w, we give an explicit lower bound and prove a matching upper bound for all networks. The special case of $w = (1, 1, \ldots, 1)$ — where the worst-case POS is the kth Harmonic number — is one of the main results in Anshelevich et al. [2]. Similarly, for atomic selfish routing games, we give tight bounds on the worst-case price of anarchy (POA) — the ratio between the cost of the worst PNE and of an optimal outcome — with respect to every set of convex cost functions (for a worst-case set of player weights). This worst-case POA is larger, but only slightly, than the one in weighted congestion games with proportional cost sharing that have PNE.

2 Preliminaries

We first explain the weighted Shapley value in general [12,23]. Consider a set S of players and a cost function $C : 2^S \to \mathbb{R}$. (For us, S is the users of a resource and $C(T)$ is the joint cost that would be incurred if the players of $T \subseteq S$ were its sole users.) For a given ordering π of the players, let $\Delta_i(\pi)$ denote $C(S^i(\pi) \cup \{i\}) - C(S^i(\pi))$, where $S^i(\pi)$ denotes the players preceding i in π.

Each player has a weight w_i and a sampling probability λ_i. Traditionally, [12,23] λ_i is set to $1/w_i$. We use the λ_i's to define a distribution over orderings of players, as follows. (When all λ_i's are equal, we get the uniform distribution and recover the usual Shapley value.) We first choose the final player in the ordering, with probabilities proportional to the λ_i's; given this choice, we choose the penultimate player randomly from the remaining ones, again with probabilities proportional to the λ_i's; and so on. The weighted Shapley value of player i is defined as the expected value of $\Delta_i(\pi)$ with respect to this distribution over orderings π.

2.1 Weighted Network Cost-Sharing

In weighted network cost-sharing, each player $i = 1, 2, \ldots, k$ has a weight $w_i \geq 1$ and a sampling weight $\lambda_i = 1/w_i$. Player i aims to construct a path P_i from a given node s_i to a given node t_i in a directed graph $G = (V, E)$, where every $e \in E$ has a fixed nonnegative cost γ_e. We next give a probabilistic representation of weighted Shapley cost shares and the corresponding potential function, in terms of independent random variables $X_1, \ldots, X_k$, where X_i is exponentially distributed with rate λ_i. With S_e the players on edge e, the weighted Shapley value of player $i \in S_e$ is defined as $\xi_i(S_e) \cdot \gamma_e$, where $\xi_i(S_e)$ is the probability that $X_i \geq X_j$ for every $j \in S_e$. Some thought shows that this definition is equivalent to the one given at the start of the section (for our joint cost function that equals 1 for every non-empty set).

A network cost-sharing game with these cost shares admits the following (weighted) potential function. For every path vector P, we define $\Phi_e(P) = \gamma_e \cdot \mathbf{E}[\max_{j \in S_e} X_j]$, where S_e is the set of players that use e in the profile P, and $\Phi(P) = \sum_{e \in E} \Phi_e(P)$. The fact that Φ is a weighted potential function can be derived easily assuming results of Hart and Mas-Colell [11] about cooperative games; but for the present cost function we can give a direct proof.

Proposition 1. *Given a path vector P, suppose player i deviates from P_i to P_i'. Let $\Delta\Phi$ be the resulting change in the potential function and ΔC_i the resulting change in the cost of player i. Then $\Delta\Phi = w_i \cdot \Delta C_i$.*

Proof. We prove the equality for each $e \in E$. By symmetry we can assume that $e \in P_i' \setminus P_i$. The cost share of player i on edge e increases from 0 to $\gamma_e \cdot \xi_i(S_e \cup \{i\})$. Let $\Delta\Phi_e$ be the change in the potential on edge e. We have

$$\Delta\Phi_e = \gamma_e \cdot \mathbf{E}\left[\left(X_i - \max_{j \in S_e} X_j\right) \middle| X_i \geq \max_{j \in S_e} X_j\right] \cdot \mathbf{Pr}\left[X_i \geq \max_{j \in S_e} X_j\right].$$

From the facts that the exponential distribution is memoryless and $w_i = 1/\lambda_i$ we get that the above is equal to $\gamma_e \cdot w_i \cdot \xi_i(S_e \cup \{i\})$. □

2.2 Atomic Selfish Routing

In atomic selfish routing, each player $i = 1, 2, \ldots, k$ has weight w_i and selects a path P_i from a node s_i to a node t_i in a given graph $G = (V, E)$. For every $e \in E$, the cost function $c_e(\cdot)$ is nonnegative and nondecreasing and the players in S_e have to pay $C_e(S_e) = f_e \cdot c_e(f_e)$, where f_e is their total weight. In atomic selfish routing, the usual (unweighted) Shapley value already gives good weight-dependent cost shares, because the joint cost function C_e is asymmetric. Let $X_{i,e}$ be a random variable with value equal to the total weight of those that appear before i in a uniformly random ordering of the players on edge e. Then the Shapley value of i for e is $c_{i,e}(S_e) = \mathbf{E}[(X_{i,e} + w_i) \cdot c_e(X_{i,e} + w_i) - X_{i,e} \cdot c_e(X_{i,e})]$.

We now present the potential function of the game. With P the selected path vector and π an arbitrary ordering of the players on e, the edge potential is

$\Phi_e(P) = \sum_{i \in S_e} c_{i,e}(S_e^i(\pi))$, where $S_e^i(\pi)$ is the set of players preceding i in π. The (exact) potential function is $\Phi(P) = \sum_{e \in E} \Phi_e(P)$. The fact that this is a potential function can be derived from Hart and Mas-Colell [11].

3 Weighted Network Cost-Sharing

3.1 Lower Bounding the Price of Stability

Given the weight vector w, which we assume is sorted in nondecreasing order, we construct the following network. Each player $i = 1, 2, \ldots, k$ starts from node s_i and they all have a common terminal node t. The network offers two possible paths to each i. The first path consists of the edge $\{s_i, v\}$ with $\gamma_{\{s_i,v\}} = 0$ and the edge $\{v, t\}$ with $\gamma_{\{v,t\}} = 1 + \epsilon$ for $\epsilon > 0$ a very small number. The second path consists of a single edge from s_i to t with $\gamma_{\{s_i,t\}} = \xi_i(\{1, 2, \ldots, i\})$. The largest player that uses its two-hop path always has an incentive to deviate to its one-hop path. Hence, in the unique PNE all players are using their one-hop paths. This implies that the worst-case POS in all games with weight vector w is lower bounded by $\sum_i \xi_i(\{1, 2, \ldots, i\})$. If we take $w = (1, 1, ..., 1)$, we recover the well-known unweighted lower bound example of Anshelevich et al. [2].

3.2 Upper Bounding the Price of Stability

We prove that for *every* weight vector w, the example of Section 3.1 is the worst case among all possible networks.

Theorem 1. *Given k players with weights $w_1 \leq w_2 \leq \ldots \leq w_k$, the worst case POS among all games with these player weights is $\sum_i \xi_i(\{1, 2, \ldots, i\})$.*

Consider a game with player set $S = \{1, 2, \ldots, k\}$ and weight vector w on $G = (V, E)$. Let P^* be the path vector that minimizes the total cost and let P be the vector of paths that minimizes the potential function and is, therefore, a PNE. We observe the changes in the total cost as players deviate one by one from their paths in P^* to their paths in P, in the order $k, k-1, \ldots, 1$. Our proof follows the main idea and steps outlined below.

Main idea and steps of the proof. As player i deviates from path P_i^* to P_i the cost of i changes, as does the cost of each j that was or is sharing edges with i. We will prove that the sum of these changes over the k deviations will be at most $C^* \cdot (\sum_i \xi_i(\{1, 2, \ldots, i\}) - 1)$, where C^* is the total cost of path vector P^*.

The first step focuses on the impact that the deviation of each i has on every $j \neq i$. The worst case for these changes is when all deviating players depart all edges shared with others and join new unused edges. We show this in a reverse order, starting from player 1 (the last to deviate) and going to player k.

The second step also focuses on the same cost changes as the first step. We show that in the worst case, every edge used in P^* is shared by all players. By the end of this step, we get that the total cost increase of non-deviating players is at most $C^* \cdot (\sum_i \xi_i(\{1, 2, \ldots, i\}) - 1)$.

The third and final step focuses on the change in the cost of the deviating player during each round. We prove that the sum over all k rounds is nonpositive.

Proof. We define the following sets of edges in the graph. For every $S_e \subseteq S$ we will write $\kappa(S_e)$ for the edges that initially (i.e., in P^*) are shared by the players in S_e. For every $S_e \subseteq S$ and for every $i \in S_e$, we will write $\delta_i(S_e)$ for the edges that are shared by the players in S_e after the deviation of $i-1$ and by the players in $S_e \setminus \{i\}$ after the deviation of i (these are the edges that i departs from). Similarly, for every $S_e \subseteq S$ and for every $i \in S_e$, we will write $\sigma_i(S_e)$ for the edges that are shared by the players in $S_e \setminus \{i\}$ after the deviation of $i-1$ and by the players in S_e after the deviation of i (these are the edges that i joins). We will write $\gamma(F)$ for the total cost of the edges in $F \subseteq E$. We will also denote the change in the potential function due to the deviation of player i as $\Delta\Phi_i$.

Consider the deviation of player i. The change in i's cost is tracked by the change in the potential function scaled by $1/w_i$, while the change that this deviation causes to the costs of other players is tracked by the portion of $\gamma(\delta_i(S_e))$ that i pays (the other players have to pay for it after i departs) and by the portion of $\gamma(\sigma_i(S_e))$ that i pays (the other players no longer have to pay for it after i joins the edges) for all $S_e \subseteq S$. So the difference between the optimal cost and the total cost of the potential function minimizer can be written as

$$\Delta C = \sum_i \frac{\Delta\Phi_i}{w_i} + \sum_i \sum_{S_e \ni i, |S_e| \geq 2} \gamma(\delta_i(S_e)) \cdot \xi_i(S_e) - \gamma(\sigma_i(S_e)) \cdot \xi_i(S_e). \quad (1)$$

At the moment i is about to deviate, the set of edges that are shared by the players in $S_e \subseteq S$ has been shaped by the deviations of $k, k-1, k-2, \ldots, i+1$ and is

$$\tau_i(S_e) = \left[\kappa(S_e) \cup \left(\bigcup_{j \in S_e, j>i} \sigma_j(S_e) \right) \cup \left(\bigcup_{j \notin S_e, j>i} \delta_j(S_e \cup \{j\}) \right) \right] \setminus \left[\left(\bigcup_{j \in S_e, j>i} \delta_j(S_e) \right) \cup \left(\bigcup_{j \notin S_e, j>i} \sigma_j(S_e \cup \{j\}) \right) \right]. \quad (2)$$

It is clear that $\delta_i(S_e) \subseteq \tau_i(S_e)$. We proceed with the following proposition.

Proposition 2. *We can obtain an upper bound for the expression in (1) by setting $\delta_i(S_e) = \tau_i(S_e)$ and $\sigma_i(S_e) = \emptyset$ for every $S_e \subseteq S$ and every $i \in S$.*

Proof. We will prove this by induction. Our base case is player 1, the last one to deviate from P_1^* to P_1. It is obvious that once everyone else has deviated, the worst case for the expression in (1) is if for every $S_e \subseteq S$ with $|S_e| \geq 2$, $\delta_1(S_e)$ is as large as possible, which means equal to $\tau_1(S_e)$, while $\sigma_1(S_e)$ is empty. Our inductive hypothesis assumes the worst case is when every player $j = 1, \ldots, n$ sets for every $S_e \subseteq S$ with $|S_e| \geq 2$, $\delta_i(S_e) = \tau_i(S_e)$ and $\sigma_i(S_e) = \emptyset$. We will prove that in the worst case, for every $S_e \subseteq S$ with $|S_e| \geq 2$, we

have $\delta_{n+1}(S_e) = \tau_{n+1}(S_e)$ and $\sigma_{n+1}(S_e) = \emptyset$. After plugging the worst case deviations for players $1, 2, \ldots, n$ into (1) using (2), we look at the coefficient of $\gamma(\delta_{n+1}(S_e))$ and the coefficient of $\gamma(\sigma_{n+1}(S_e))$, for which we write $\alpha(\delta_{n+1}(S_e))$ and $\alpha(\sigma_{n+1}(S_e))$ respectively. So, from (1), (2), and the inductive hypothesis, we get that for every $S_e \subseteq S$,

$$\begin{aligned}\alpha(\delta_{n+1}(S_e)) &= -\alpha(\sigma_{n+1}(S_e)) \\ &= \xi_{n+1}(S_e) + \sum_{j \in S_e, j<n+1} \alpha(\delta_j(S_e \setminus \{n+1\})) - \alpha(\delta_j(S_e)).\end{aligned}$$

To complete the proof of the proposition, it suffices to show that this is nonnegative. Assume $S_e = \{i_1, i_2, \ldots, i_m\}$ in nondecreasing weight order. The solution of this recurrence is as follows.

$$\alpha(\delta_{i_m}(S_e)) = \sum_{j=2}^{m} \xi_j(\{i_1, \ldots, i_j\}) - \sum_{j=1}^{m-1} \xi_j(\{i_1, \ldots, i_j, i_m\}) \geq 0, \text{ and for } l < m,$$

$$\alpha(\delta_{i_l}(S_e)) = \sum_{j=1}^{l} \xi_j(\{i_1, \ldots, i_j, i_{l+1}, \ldots, i_m\}) - \sum_{j=1}^{l-1} \xi_j(\{i_1, \ldots, i_j, i_l, \ldots, i_m\}) \geq 0.$$

□

From (1) and Proposition 2, we get that

$$\Delta C \leq \sum_i \frac{\Delta\Phi_i}{w_i} + \sum_i \sum_{S_e \ni i, |S_e| \geq 2} \gamma(\tau_i(S)) \cdot \xi_i(S).$$

We order the players in $S_e \subseteq S$ in nondecreasing weight order as $i_1, i_2, \ldots, i_m$ and we denote $\Xi(S_e) = \sum_{j=1}^{m} \xi_j(\{i_1, i_2, \ldots, i_j\})$. Then, we can see that the inequality above becomes

$$\Delta C \leq \sum_i \frac{\Delta\Phi_i}{w_i} + \sum_{S_e \subseteq S} \gamma(\kappa(S_e)) \cdot [\Xi(S_e) - 1]. \tag{3}$$

We can see this as follows. The cost $\gamma(\kappa(S_e))$ appears with coefficient $\xi_m(S_e)$ when i_m departs from all these edges, then with coefficient $\xi_{m-1}(S_e \setminus \{i_m\})$ when i_{m-1} departs from all these edges, etc. We proceed with the following proposition.

Proposition 3. *For any set of players S we have* $\max_{S_e \subseteq S} \Xi(S_e) = \Xi(S)$.

Proof. It suffices to show that for any player $j \in S$ it is the case that $\Xi(S) \geq \Xi(S \setminus \{j\})$. Recall that X_i is an exponential random variable with parameter $\lambda_i = 1/w_i$ and that the players in S are such that $1 \geq \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_k$. We will write $B_i(S_e)$ for the event that X_i is larger than X_j for every $j \in S_e$. We can see that $\Xi(S) = 1 + \sum_{i=2}^{k} \mathbf{Pr}[B_i(\{1, 2, \ldots, i\})]$ and

$$\Xi(S \setminus \{j\}) = 1 + \sum_{i=2}^{j-1} \mathbf{Pr}[B_i(\{1, 2, \ldots, i-1\})]$$

$$+ \sum_{i=j+1}^{k} \mathbf{Pr}[B_i(\{1,2,\ldots,i-1\} \setminus \{j\})].$$

Hence, we get

$$\begin{aligned}
&\Xi(S) - \Xi(S \setminus \{j\}) \\
&= \mathbf{Pr}[B_j(\{1,2,\ldots,j-1\})] - \sum_{i=j+1}^{k} \mathbf{Pr}[B_i(\{1,2,\ldots,i-1\} \setminus \{j\}) \wedge B_j(\{i\})] \\
&= \int_0^\infty \lambda_j e^{-\lambda_j x} \prod_{l=1}^{j-1}(1-e^{-\lambda_l x}) - \sum_{i=j+1}^{k} \lambda_i e^{-\lambda_i x} e^{-\lambda_j x} \prod_{l=1, l\neq j}^{i-1}(1-e^{-\lambda_l x}) dx \\
&\geq \int_0^\infty \lambda_j e^{-\lambda_j x} \prod_{l=1}^{j-1}(1-e^{-\lambda_l x}) - \sum_{i=j+1}^{k} \lambda_j e^{-\lambda_j x} e^{-\lambda_j x} \prod_{l=1, l\neq j}^{i-1}(1-e^{-\lambda_l x}) dx \\
&\geq \int_0^\infty \lambda_j e^{-\lambda_j x} \prod_{l=1}^{j-1}(1-e^{-\lambda_l x}) \left[1 - \sum_{i=j+1}^{k} e^{-\lambda_j x}(1-e^{-\lambda_l x})^{i-j-1}\right] dx \geq 0. \quad \square
\end{aligned}$$

From (3) and Proposition 3, we get

$$\Delta C \leq \sum_i \frac{\Delta\Phi_i}{w_i} + C^* \cdot [\Xi(S) - 1], \tag{4}$$

with C^* the optimal total cost. At this point, to complete the proof of the theorem, we only need to show that the following proposition holds.

Proposition 4. *It is the case that* $\sum_{i=1}^{k} \Delta\Phi_i / w_i \leq 0$.

Proof. We will prove by induction that for every $j = 2, 3, \ldots, k-1$ it is the case that

$$\sum_{i=1}^{j} \frac{\Delta\Phi_i}{w_i} \leq \frac{\sum_{i=1}^{j} \Delta\Phi_i}{w_j}.$$

Our base is the case with $j = 2$. Note that $\Delta\Phi_1$ is the difference in the potential function when moving from some path vector to the potential function minimizer, so it is nonpositive. We get

$$\frac{\Delta\Phi_1}{w_1} + \frac{\Delta\Phi_1}{w_2} \leq \frac{\Delta\Phi_1 + \Delta\Phi_2}{w_2}.$$

Our inductive hypothesis assumes the statement is true for $j = n$ and we will prove it is true for $j = n+1$. We have

$$\sum_{i=1}^{n+1} \frac{\Delta\Phi_i}{w_i} \leq \frac{\Delta\Phi_{n+1}}{w_{n+1}} + \frac{\sum_{i=1}^{n} \Delta\Phi_i}{w_n} \leq \frac{\sum_{i=1}^{n+1} \Delta\Phi_i}{w_{n+1}}.$$

The last step follows from the fact that $\sum_{i=1}^{n} \Delta\Phi_i$ is the change in the value of the potential function while moving from some path vector to the potential function minimizer and is, therefore, nonpositive. Now we have

$$\sum_{i=1}^{k} \frac{\Delta\Phi_i}{w_i} \leq \frac{\sum_{\mathrm{i=1}}^{\mathrm{k}} \Delta\Phi_i}{w_k} \leq 0,$$

which proves the proposition. □

Combining Proposition 4 with (4) proves the theorem. □

This result implies that as long as all players' weights lie in a bounded range, the price of stability remains $O(\log k)$ (unlike with proportional cost sharing [5]).

3.3 Further Analysis of the Price of Stability

In this subsection we focus on the expression $\sum_i \xi_i(\{1, 2, \ldots, i\})$, which is the worst case POS for a game with weight vector w. We first prove the following useful lemma.

Lemma 1. *The value $\xi_i(\{1, 2, \ldots, i\})$ does not decrease if we replace the weights of players $1, 2, \ldots, i-1$ with their average.*

Proof. We can assume that $w_{i-1} > w_1$, and show that the cost share of player i increases if we replace w_{i-1} with $w_{i-1} - \epsilon$ and w_1 with $w_1 + \epsilon$ for a very small ϵ. Recall from Section 2 our probabilistic representation of the weighted Shapley cost shares. Suppose that the maximum of $X_2, X_3, \ldots, X_{i-2}$ is y. Consider the following experiment to determine the values of X_1, X_{i-1}. Numbers η_1 and η_{i-1} are picked uniformly and independently at random from $[0, 1]$. The value of X_1 is $F_{\lambda_1}^{-1}(\eta_1) = -w_1 \ln(1 - \eta_1)$, which is the inverse of the cumulative distribution function of the exponential distribution with parameter $\lambda_1 = 1/w_1$. This does not change the distribution of X_1. Similarly the value of X_{i-1} is $-w_{i-1} \ln(1 - \eta_{i-1})$.

For the same η_1 and η_{i-1}, after we have changed the weights, the values become $-(w_1+\epsilon)\ln(1-\eta_1)$ and $-(w_{i-1}-\epsilon)\ln(1-\eta_1)$. We know that $\xi_i(\{1, 2, \ldots, i\})$ is the probability that X_i is larger than all X_j for $j = 1, 2, \ldots, i-1$. Hence, to prove the lemma, it suffices to show that this probability increases with the updated weights. The (η_1, η_{i-1}) pairs which make it harder for X_i to be the largest with the updated weights compared to the original weights, are the ones where X_1 is larger than X_{i-1} and also larger than y. We assume ϵ is small enough to keep the relative ordering of $X_1, X_2, \ldots, X_{i-1}$ intact even after the change of the weights. It is clear that such ϵ exists. We prove that for every pair (η, η') that decreases the probability of X_i being the largest by Δp, the pair (η', η) increases the same probability by $\Delta p' \geq \Delta p$. We have

$$\Delta p = e^{\lambda_i w_1 \ln(1-\eta)} - e^{\lambda_i (w_1+\epsilon) \ln(1-\eta)} = \left[(1-\eta)^{\lambda_i}\right]^{w_1} - \left[(1-\eta)^{\lambda_i}\right]^{w_1+\epsilon}.$$

Since X_1, which has a distribution with larger rate, is bigger than X_{i-1} and y when the pair (η, η') is picked, it follows that X_{i-1} is larger than X_1 and y when

the pair (η', η) is picked. We then have

$$\Delta p' = e^{\lambda_i (w_{i-1}-\epsilon)\ln(1-\eta)} - e^{\lambda_i w_{i-1}\ln(1-\eta)} = \left[(1-\eta)^{\lambda_i}\right]^{w_{i-1}-\epsilon} - \left[(1-\eta)^{\lambda_i}\right]^{w_{i-1}}.$$

Since $\left[(1-\eta)^{\lambda_i}\right]^x$ is decreasing and convex for $x > 0$, we get $\Delta p' \geq \Delta p$, which completes the proof. □

We can now prove the following upper bound for $\xi_i(\{1, 2, \ldots, i\})$.

Lemma 2. *Let $S_e = \{1, 2, \ldots, i\}$. It is the case that*

$$\xi_i(S_e) \leq \left(\frac{w_i' + 1}{w_i' i + 1}\right)^{1/w_i'}$$

where w_i' is the ratio of w_i to the average weight of payers $1, 2, \ldots, i-1$.

Proof. From Lemma 1 we get that we can obtain an upper bound on $\xi_i(S_e)$ by substituting every w_i for $i = 1, 2, \ldots, i-1$ with their average, which we denote $\bar{w}_{-i}$. This would be equivalent to having a weight $w_i' = w_i/\bar{w}_{-i}$ for player i and weight 1 for all other $i-1$ players. Then using the definition of the weighted Shapley value from the start of Section 2, we get

$$\begin{aligned}\xi_i(S_e) &\leq \prod_{j=1}^{i-1} \frac{jw_i'}{jw_i'+1} = \prod_{j=1}^{i-1}\left(1 - \frac{1}{jw_i'+1}\right) \leq \prod_{j=1}^{i-1} e^{-1/(jw_i'+1)} \\ &= \exp\left(-\sum_{j=1}^{i-1}\frac{1}{jw_i'+1}\right) \leq \exp\left(-\int_1^i \frac{dx}{xw_i'+1}\right) = \left(\frac{w_i'+1}{w_i' i+1}\right)^{1/w_i'}.\end{aligned}$$

This completes the proof. □

Combining Lemma 2 with Theorem 1, provides an expression that upper bounds the POS for a given weight vector. An example is the case with equal weights, where we get the tight $H_k = O(\log(k))$ bound shown in [2]. Another interesting case is that with weight $w_i = i$ for $i = 1, 2, \ldots, k$, for which we get that the POS is $O(\sqrt{k})$.

4 Atomic Selfish Routing

Let $\mathcal{C}$ be a set of nonnegative nondecreasing cost functions and suppose all resource cost functions of our atomic selfish routing game are picked from $\mathcal{C}$. We make the following assumptions for every $c \in \mathcal{C}$ and $w \geq 0$. It is the case that $(x+w)\cdot c(x+w) - x \cdot c(x)$ is a convex and nondecreasing function of x (this is true, for example, for twice differentiable functions with nondecreasing first and second derivatives). Also, if $\bar{c}(x) = w \cdot c(x)$, then $\bar{c} \in \mathcal{C}$ (i.e., $\mathcal{C}$ is closed under scaling). Finally, if $\bar{c}(x) = c(w \cdot x)$, then $\bar{c} \in \mathcal{C}$ (i.e., $\mathcal{C}$ is closed under dilation).

4.1 Upper Bounding the Price of Anarchy

We define

$$\mathcal{A}(\mathcal{C}) = \Bigg\{(\lambda,\mu) : \mu < 1, \quad \text{for all} \quad x, x^* \geq 0 \quad \text{and for all} \quad c \in \mathcal{C}$$
$$\left(\lambda - \frac{1}{2}\right) x^* \cdot c(x^*) + \left(\mu + \frac{1}{2}\right) x \cdot c(x) - \frac{1}{2}(x + x^*) \cdot c(x + x^*) \geq 0\Bigg\}.$$

The following proposition gives an upper bound on the POA.

Proposition 5. *If $(\lambda,\mu) \in \mathcal{A}(\mathcal{C})$, then the POA of an atomic selfish routing game with Shapley cost shares and cost functions from $\mathcal{C}$ is at most $\lambda/(1-\mu)$.*

Using the above proposition we derive the upper bound $\zeta(\mathcal{C}) = \inf\{\lambda/(1-\mu) : (\lambda,\mu) \in \mathcal{A}(\mathcal{C})\}$. This bound is robust in the sense of [21], and thus applies also to more general equilibrium concepts, like coarse correlated equilibria. If $\mathcal{A}(\mathcal{C})$ is empty, then we define $\zeta(\mathcal{C}) = \infty$.

Upper Bound for Polynomials. Suppose $\mathcal{C}$ is the class of polynomials with nonnegative coefficients and maximum degree d. Also let χ_d be the number that satisfies $3\chi_d^{d+1} = 1 + (\chi_d + 1)^{d+1}$. We get the following theorem.

Theorem 2. *If $\mathcal{C}$ is the set of polynomials with nonnegative coefficients and maximum degree d, then the POA of an atomic selfish routing game with Shapley cost shares and cost functions in $\mathcal{C}$ is at most $\chi_d^{d+1} = (\Theta(d))^{d+1}$.*

For comparison, the worst-case POA with proportional sharing, in such games that happen to possess PNE, is the slightly smaller quantity $\Theta((d/\ln d)^{d+1})$.

4.2 Lower Bounding the Price of Anarchy

The upper bounds presented in Subsection 4.1 are tight. The analysis that proves this is similar to the one for weighted congestion games with proportional cost sharing [4].

Lower Bound for Polynomials. We now present the lower bound construction for the case when $\mathcal{C}$ is the set of polynomials with nonnegative coefficients and maximum degree d. Consider a game with the set of players being $S = \{1, 2, \ldots, k\}$ and the set of edges being $E = \{1, \ldots, k+1\}$. The weight of player $i \in S$ is $w_i = \chi_d^i$, the cost function of edge $k+1$ is $c_{k+1}(x) = x^d$, and the cost function of every other $e \in E$ is $c_e(x) = \chi_d^{(d+1)(k-e)} x^d$. Each player i has to select between using edge i and edge $i+1$. The fact that the outcome where every player i picks edge i is a PNE gives the following theorem.

Theorem 3. *The POA of an atomic selfish routing game with Shapley cost shares and cost functions that are polynomials with nonnegative coefficients and maximum degree d can be arbitrarily close to χ_d^{d+1}.*

References

1. Aland, S., Dumrauf, D., Gairing, M., Monien, B., Schoppmann, F.: Exact price of anarchy for polynomial congestion games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 218–229. Springer, Heidelberg (2006)
2. Anshelevich, E., Dasgupta, A., Kleinberg, J., Tardos, É., Wexler, T., Roughgarden, T.: The price of stability for network design with fair cost allocation. SIAM Journal on Computing 38(4), 1602–1623 (2008)
3. Awerbuch, B., Azar, Y., Epstein, A.: The price of routing unsplittable flow. In: STOC, pp. 57–66 (2005)
4. Bhawalkar, K., Gairing, M., Roughgarden, T.: Weighted congestion games: Price of anarchy, universal worst-case examples, and tightness. In: de Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6347, pp. 17–28. Springer, Heidelberg (2010)
5. Chen, H., Roughgarden, T.: Network design with weighted players. Theory of Computing Systems 45(2), 302–324 (2009)
6. Chen, H., Roughgarden, T., Valiant, G.: Designing network protocols for good equilibria. SIAM Journal on Computing 39(5), 1799–1832 (2010)
7. Fotakis, D., Kontogiannis, S.C., Spirakis, P.G.: Selfish unsplittable flows. Theoretical Computer Science 348(2-3), 226–239 (2005)
8. Gairing, M., Schoppmann, F.: Total latency in singleton congestion games. In: Deng, X., Graham, F.C. (eds.) WINE 2007. LNCS, vol. 4858, pp. 381–387. Springer, Heidelberg (2007)
9. Goemans, M.X., Mirrokni, V., Vetta, A.: Sink equilibria and convergence. In: FOCS, pp. 142–151 (2005)
10. Harks, T., Klimm, M.: On the existence of pure Nash equilibria in weighted congestion games. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 79–89. Springer, Heidelberg (2010)
11. Hart, S., Mas-Colell, A.: Potential, value, and consistency. Econometrica 57(3), 589–614 (1989)
12. Kalai, E., Samet, D.: On weighted Shapley values. International Journal of Game Theory 16(3), 205–222 (1987)
13. Milchtaich, I.: Congestion games with player-specific payoff functions. Games and Economic Behavior 13(1), 111–124 (1996)
14. Monderer, D., Shapley, L.S.: Fictitious play property for games with identical interests. Journal of Economic Theory 68, 258–265 (1996)
15. Monderer, D., Shapley, L.S.: Potential games. Games and Economic Behavior 14(1), 124–143 (1996)
16. Mosk-Aoyama, D., Roughgarden, T.: Worst-case efficiency analysis of queueing disciplines. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 546–557. Springer, Heidelberg (2009)
17. Moulin, H.: The price of anarchy of serial, average and incremental cost sharing. Economic Theory 36(3), 379–405 (2008)
18. Osborne, M.J., Rubinstein, A.: A Course in Game Theory. MIT Press, Cambridge (1994)
19. Rosenthal, R.W.: A class of games possessing pure-strategy Nash equilibria. International Journal of Game Theory 2(1), 65–67 (1973)

20. Rosenthal, R.W.: The network equilibrium problem in integers. Networks 3(1), 53–59 (1973)
21. Roughgarden, T.: Intrinsic robustness of the price of anarchy. In: STOC, pp. 513–522 (2009)
22. Roughgarden, T., Tardos, É.: How bad is selfish routing? Journal of the ACM 49(2), 236–259 (2002)
23. Shapley, L.S.: Additive and Non-Additive Set Functions. PhD thesis, Department of Mathematics, Princeton University (1953)
24. Shenker, S.J.: Making greed work in networks: A game-theoretic analysis of switch service disciplines. IEEE/ACM Transactions on Networking 3(6), 819–831 (1995)

Existence and Uniqueness of Equilibria for Flows over Time*

Roberto Cominetti, José R. Correa, and Omar Larré

Department of Industrial Engineering, Universidad de Chile, Santiago, Chile

Abstract. Network flows that vary over time arise naturally when modeling rapidly evolving systems such as the Internet. In this paper, we continue the study of equilibria for flows over time in the single-source single-sink deterministic queuing model proposed by Koch and Skutella. We give a constructive proof for the existence and uniqueness of equilibria for the case of a piecewise constant inflow rate, through a detailed analysis of the static flows obtained as derivatives of a dynamic equilibrium.

Keywords: Flows Over Time, Network Equilibrium.

1 Introduction

Understanding time varying flows in a network is fundamental in contexts where a *steady state* is rarely attained, including traffic networks and the Internet. Furthermore, these systems are usually characterized by the lack of coordination among participating agents and therefore need to be considered from a game theoretic perspective.

Research in flows over time has mainly focused in optimization. The first to consider this model in a discrete time setting were Ford and Fulkerson [6,7] who designed an algorithm to find a flow over time carrying the maximum possible total flow in a given time horizon. Gale [8] then showed the existence of a flow pattern that achieves this optimum simultaneously for all time horizons. These results were extended to a continuous time setting by Fleischer and Tardos [5] and by Anderson and Philpott [1] respectively. We refer the reader to Skutella [12] for an excellent survey.

The study of network flows over time when different flow particles act selfishly has mostly been considered in the transportation science literature. The book of Ran and Boyce [11] describes a very general model of equilibria, for which unfortunately very little is known. More recently, Koch and Skutella [9] considered a simpler model in which there is an inflow rate at a single source node that travels across the network towards a single sink node through edges that are characterized by their travel time (or latency) and their *per-time-unit* capacity. The model can be interpreted as a fluid relaxation of a deterministic queuing model, and is a special case of Ran and Boyce's general framework. Very recently this model was also considered by Macko, Larson, and Steskal [10] to study Braess's type paradoxes, and by Bhaskar, Fleischer, and Anshelevich [2] to design an effective stackleberg routing strategy.

* Supported in part by Instituto Milenio Sistemas Complejos de Ingeniería, and FONDECYT grants 1090050 and 1100046.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 552–563, 2011.

Our Contribution. Building upon the work of Koch and Skutella, in this paper we give a constructive (algorithmic) proof for the existence and uniqueness of equilibria for flows over time. Our work is based on the key concept of *thin flow with resetting* introduced by Koch and Skutella. This concept defines a static flow together with an associated labeling which coincides with the derivatives of the distance labels of an equilibrium. We actually consider a slightly more restrictive definition by adding an additional technical constraint, and show how to transform any thin flow with resetting into a flow and labeling satisfying this extra property. Then we prove that such a flow always exists using an equivalent nonlinear complementarity problem which is shown to admit a solution by proving that an associated variational inequality has an interior solution. Furthermore, we prove that a thin flow with resetting with our additional condition is also unique. By integrating these thin flows with resetting, the results allow us to prove the existence and uniqueness (among a natural family) of equilibria for flows over time when the inflow rate is piecewise constant. Finally, we give a non-constructive existence proof when the inflow rate function is measurable and its value raised to the p-th power has finite integral, for some $1 < p < \infty$ (i.e., belongs to L^p).

As a by-product of our existence result for thin flows with resetting, we obtain that the problem of finding such a flow and labeling belongs to TFNP (since a solution is guaranteed to exist). Furthermore our proof strategy shows that the problem belongs to PPAD as it reduces to finding a fixed point. In any case, we conjecture that the problem is actually solvable in polynomial time.

Organization of the paper. In §2, we describe the equilibrium model for flows over time introduced by Koch and Skutella [9]: First, we present the dynamic flow model, and then we define the routing game and its equilibria. In §3 we prove our main results concerning the existence and uniqueness of our refined definition of thin flow with resetting. Finally, in §4 we discuss how the latter results apply to equilibria for flows over time. Due to space limitations, several proofs are deferred to the full version.

2 A Model for Dynamic Routing Games

We consider a network (G, u, τ, s, t) consisting of a directed graph $G = (V, E)$ with node set V and edge set E, a vector $u = (u_e)_{e \in E}$ of strictly positive real numbers representing edges capacities, a vector $\tau = (\tau_e)_{e \in E}$ of nonnegative real numbers representing free flow transit times, a source $s \in V$, and a sink $t \in V$. An edge $e \in E$ from node v to node w is denoted (v, w) or just vw. To avoid confusion, we assume without loss of generality that there is at most one edge between any pair of nodes in G and that there are no loops. We also assume that for every cycle C of G, the sum of the free flow transit times along C is not zero, i.e.,

$$\sum_{e \in C} \tau_e > 0, \quad \text{for every cycle } C \text{ of } G. \tag{1}$$

2.1 Flows over Time

A *flow over time* is a pair of arrays $f := (f^+, f^-)$ of Lebesgue-integrable functions whose components $f_e^+ : \mathbb{R}_+ \to \mathbb{R}_+$ and $f_e^- : \mathbb{R}_+ \to \mathbb{R}_+$ for each $e \in E$ represent

respectively the rate at which flow enters the tail of e and the rate of flow leaving the head of e at each time θ. For every edge $e \in E$ we define the cumulative inflow $F_e^+(\theta)$, the cumulative outflow $F_e^-(\theta)$, and the queue size $z_e(\theta)$ at time $\theta \geq 0$ as

$$F_e^+(\theta) := \int_0^\theta f_e^+(\xi)d\xi, \quad F_e^-(\theta) := \int_0^\theta f_e^-(\xi)d\xi, \quad z_e(\theta) := F_e^+(\theta) - F_e^-(\theta + \tau_e).$$

Definition 1. *A flow over time $f = (f^+, f^-)$ is said to be* feasible *if and only if it satisfies for almost all $\theta \geq 0$:*

1. *the* capacity constraints $f_e^-(\theta) \leq u_e$ *for each* $e \in E$
2. *the* non-deficit constraints $z_e(\theta) \geq 0$ *for each* $e \in E$
3. *the* flow conservation constraints

$$\sum_{e \in \delta^-(v)} f_e^-(\theta) = \sum_{e \in \delta^+(v)} f_e^+(\theta) \quad \textit{for each } v \in V \setminus \{s, t\}. \tag{2}$$

We also define the *network inflow rate* (at node s) as

$$u_0(\theta) = \sum_{e \in \delta^+(s)} f_e^+(\theta) - \sum_{e \in \delta^-(s)} f_e^-(\theta),$$

and the *waiting time at edge* e at time θ by

$$q_e(\theta) := \frac{z_e(\theta)}{u_e} = \frac{F_e^+(\theta) - F^-(\theta + \tau_e)}{u_e} \tag{3}$$

so that the mapping $\theta \mapsto \theta + q_e(\theta)$ represents the time at which a flow particle entering e at time θ exits from the queue. Koch and Skutella [9] showed the following.

Proposition 1. *The function $\theta \mapsto \theta + q_e(\theta)$ is nondecreasing and continuous.*

2.2 Label Functions for Flows over Time

For each $w \in V$ let $\mathcal{P}_w$ denote the set of all s-w paths (not necessarily simple) in G. Given an s-t flow over time, a node $w \in V$ and a path $P = (e_1, e_2, ..., e_k) \in \mathcal{P}_w$, with a corresponding sequence of nodes $(v_1, v_2, ..., v_{k+1})$ (*i.e.*, $e_i = v_i v_{i+1}$, $v_1 = s$ and $v_{k+1} = w$), we define $\ell_{w,P} : \mathbb{R}_+ \to \mathbb{R}_+$, the *label function of node w using path P*, by

$$\begin{aligned} \ell_{s,P}(\theta) &:= \theta, \\ \ell_{v_{i+1},P}(\theta) &:= \ell_{v_i,P}(\theta) + \tau_{e_i} + q_e(\ell_{v_i,P}(\theta)), \text{ for all } i \in \{1, ..., k\}. \end{aligned}$$

Thus $\ell_{w,P}(\theta)$ represents the time a flow particle reaches w traveling through P, starting from s at time θ. Note that these label functions depend on the full flow over time f through the waiting time functions q_e. We may then define the minimum time at which a flow particle can reach a node as follows (as we shall see, this definition is consistent with that of Koch and Skutella [9]).

Definition 2 (Label functions). *Given a feasible s-t flow over time f we define for every node $v \in V$ the label functions $\ell_v : \mathbb{R}_+ \to \mathbb{R}_+$ as*

$$\ell_v(\theta) := \min_{P \in \mathcal{P}_v} \ell_{v,P}(\theta). \tag{4}$$

Naturally, a particle traveling from s to w in the shortest possible time will not use a path containing a cycle. This is indeed a consequence of the non-decreasing monotonicity of the mapping $\theta \mapsto \theta + q_e(\theta)$, $q_e \geq 0$ and $\tau_e \geq 0$, for all $e \in E$, so that the minimum $\ell_{w,P}(\theta)$ for $P \in \mathcal{P}_w$ is attained over simple paths. Also, as a consequence of the non-decreasing monotonicity and continuity of the map $\theta \mapsto \theta + q_e(\theta)$, we have that for each node $v \in V$, the label function ℓ_v is non-decreasing and continuous. In addition, the functions $(\ell_v)_{v \in V}$ satisfy the definition in [9], which states that

$$\ell_w(\theta) = \begin{cases} \theta & \text{for } w = s \\ \min\limits_{e=vw \in \delta^-(w)} \{\ell_v(\theta) + \tau_e + q_e(\ell_v(\theta))\} & \text{for } w \in V \setminus \{s\}. \end{cases}$$

2.3 Equilibria

A feasible s-t flow over time $f = (f^+, f^-)$ on a network (G, u, τ, s, t), can be seen as a *dynamic equilibrium* if we identify each flow particle that travels from s to t at a time $\theta \in \mathbb{R}_+$ as players. In a dynamic equilibrium, all flow particles travel on s-t paths that yield the shortest possible total travel time, taking into account the aggregate congestion generated within the network due to other particles. For a formal definition we refer to [9]. In what follows we rely on an equivalent characterization given in Theorem 1 below.

Definition 3. *We say that edge $e \in E$ is contained in a shortest path at time $\theta \geq 0$ if and only if $\ell_w(\theta) = \ell_v(\theta) + \tau_e + q_e(\ell_v(\theta))$. We denote E_θ the set of all such arcs and we consider the induced acyclic graph $G_\theta = (V, E_\theta)$ which we call the* ***current shortest paths network*** *. We also say that* ***flow is sent along currently shortest paths*** *if, for each edge $e = vw \in E$, the following holds for almost all $\theta \geq 0$*

$$f_e^+(\ell_v(\theta)) > 0 \Longrightarrow e \in E_\theta.$$

In other words, a flow that is sent along currently shortest paths is such that at time $\ell_v(\theta)$ there is no flow assigned to any edge $e = vw \in E$ not lying in a shortest path at time θ. Note that there are edges in G not contained in a current shortest path network. Also, since (G, u, τ, s, t) satisfies Condition (1), G_θ is acyclic for every $\theta \geq 0$.

Following Koch and Skutella, let us consider a particle at s at time θ, $x_e^+(\theta)$ the amount of flow assigned to $e = vw$ before this particle can reach v, and $x_e^-(\theta)$ the flow leaving $e = vw$ before this particle can reach w, that is:

$$x_e^+(\theta) := F_e^+(\ell_v(\theta)), \quad x_e^-(\theta) := F_e^-(\ell_w(\theta)).$$

Theorem 1 ([9]). *For a feasible s-t flow over time, the following are equivalent:*

(i) The given flow over time is an equilibrium.
(ii) Flow is only sent along currently shortest paths.
(iii) For each edge $e \in E$ and at all times $\theta \geq 0$, it holds that $x_e^+(\theta) = x_e^-(\theta)$.

Whenever any of the three statements above is satisfied, then $x^+ = x^-$. In that case, we define $x_e(\theta) := x_e^+(\theta)$ for all $\theta \geq 0$ and each $e \in E$. From now on, we call $(x_e(\theta))_{e \in E}$ the *underlying static flow at time θ.*

3 Derivative of an Equilibrium Flow: A Special Labeling

In this section we study a key node labeling for flows over time, proving its existence and uniqueness. This labeling arises from the derivative of the distance label functions of a flow over time at equilibrium, and can be used in order to construct such an equilibrium from its derivatives.

Definition 4 (Thin Flow with Resetting). *Let (G, u, s, t) be a network, $E_1 \subset E(G)$ a subset of edges and $F \geq 0$. A labeling $(\ell'_v)_{v\in V(G)}$ is a thin flow of value F with resetting on E_1 if there exists a static s-t F-flow $(x'_e)_{e\in E(G)} \geq 0$ (i.e., a nonnegative and conservative static flow sending an amount F from s to t) such that:*

$$\ell'_s = 1, \tag{5}$$

$$\ell'_w \leq \ell'_v \qquad \text{for all } e = vw \in E(G) \setminus E_1, x'_e = 0, \tag{6}$$

$$\ell'_w = \max\{\ell'_v, \frac{x'_e}{u_e}\} \qquad \text{for all } e = vw \in E(G) \setminus E_1, x'_e > 0, \tag{7}$$

$$\ell'_w = \frac{x'_e}{u_e} \qquad \text{for all } e = vw \in E_1, \tag{8}$$

$$\ell'_w \geq \min_{vw\in\delta^-_G(w)} \ell'_v \qquad \text{if } \delta^-_G(w) \cap E_1 = \emptyset. \tag{9}$$

The original definition of thin flow with resetting, due to Koch and Skutella [9], does not consider Condition (9). They showed that the derivatives of the label functions of a dynamic equilibrium, restricted to the current shortest paths network, satisfy Conditions (5)-(8). We introduce Condition (9), to guarantee uniqueness of thin flows with resetting, as illustrated in Example 1 below. The following theorem shows that the derivatives of the label functions of a dynamic equilibrium, restricted to the current shortest paths network, are a thin flow with resetting. This result is applicable if the derivatives of the label and the underlying static flow functions exist. Interestingly, both the label functions and the underlying static flow functions are monotonically nondecreasing implying that both families of functions are differentiable almost everywhere.

Theorem 2. *Consider a feasible s-t flow over time f which is an equilibrium on a network (G, u, τ, s, t) with corresponding label functions $(\ell_v)_{v\in V}$, network inflow rate u_0, edge waiting time functions $(q_e)_{e\in E}$ and underlying static flow $(x_e)_{e\in E}$. Then for almost all $\theta \geq 0$, $\frac{dx_e}{d\theta}(\theta)$ and $\frac{d\ell_v}{d\theta}(\theta)$ exist for each $e \in E$ and $v \in V$. Moreover, for almost all $\theta \geq 0$, on the current shortest paths network G_θ, the derivatives $\left(\frac{d\ell_v}{d\theta}(\theta)\right)_{v\in V}$ form a thin flow of value $u_0(\theta)$ with resetting on the waiting edges $E_1 := \{e = vw \in E \mid q_e(\ell_v(\theta)) > 0\}$. A corresponding static flow fulfilling (5) to (9) is given by the derivatives $\left(\frac{dx_e}{d\theta}(\theta)\right)_{e\in E}$.*

The proof, found in the full version of the paper, follows closely that of Koch and Skutella though we need to be careful when dealing with Condition (9).

Example 1. To illustrate the role of Condition (9) in the construction of an equilibrium, consider a network composed of three nodes $\{s, v, t\}$ and three edges: one from s to t, another from s to v, and the last one from v to t. Every edge e has capacity $u_e = 1$, and

the travel times are $\tau_{st} = \tau_{vt} = 0$ and $\tau_{sv} = 1$. Suppose that the network inflow rate is $u_0(\theta) = 2$ if $\theta \in [0, 1)$ and $u_0(\theta) = 1/2$ if $\theta \in [1, \infty)$. At equilibrium, in the beginning flow is sent over the edge st only, so that a queue begins to build up on edge st until time $\theta = 1$, when the waiting time is $q_{st}(1) = 1$, the shortest network path becomes the complete network and the network inflow rate changes in value to $u_0 = 1/2$. After time $\theta = 1$ flow is only sent over edge st again. It is easy to check that the label functions from $\theta = 1$ (and at least for a while) are $\ell_v(\theta) = 1 + \theta$ and $\ell_t(\theta) = 3/2 + \theta/2$, so that the derivatives (from the right) at time $\theta = 1$ are $\frac{d\ell_v}{d\theta}(1) = 1$ and $\frac{d\ell_t}{d\theta}(1) = 1/2$. Indeed, for the shortest path network at time $\theta = 1$ (the complete network), the corresponding thin flow of value $1/2$ with resetting on $E_1 = \{st\}$ is $\ell'_v = 1$ and $\ell'_t = 1/2$, but without Condition (9), the label ℓ'_v could take any value between $1/2$ and 1.

3.1 Existence of Thin Flows with Resetting

In this section we prove the existence of thin flows with resetting when the underlying graph is acyclic. This result will then be used to conclude the existence of equilibria for flows over time. To prove the existence of thin flows with resetting we show that this is equivalent to the existence of solutions of a nonlinear complementarity problem, which in turn is equivalent to finding an interior solution of a certain variational inequality.

We start with a basic result that will be needed later. Given a subset K of the Euclidean n-dimensional space $\mathbb{R}^n$ and a mapping $\Phi : K \to \mathbb{R}^n$, the variational inequality, denoted $\mathrm{VI}(K, \Phi)$, is to find a vector $x \in K$ such that

$$(y - x)^t \Phi(x) \geq 0, \quad \forall y \in K.$$

The solution set to this problem is denoted $\mathrm{SOL}(K, \Phi)$. When $K = \mathbb{R}^n_+$, then VI admits an equivalent form known as a *nonlinear complementarity problem* . Given a mapping $\Phi : \mathbb{R}^n_+ \to \mathbb{R}^n$, the nonlinear complementarity problem, denoted $\mathrm{NCP}(\Phi)$, is to find a vector $x \in \mathbb{R}^n_+$ satisfying

$$\Phi(x) \geq 0 \quad \text{and} \quad x^t \Phi(x) = 0.$$

The equivalence between $\mathrm{NCP}(\Phi)$ and $\mathrm{VI}(\mathbb{R}^n_+, \Phi)$ is easy (see *e.g.* [4]). Also, we have the following result, whose proof follows from Brouwer's fixed point theorem [4].

Theorem 3. *Let $K \subset \mathbb{R}^n$ be compact convex and let $\Phi : K \to \mathbb{R}^n$ be continuous. Then the set $\mathrm{SOL}(K, \Phi)$ is nonempty and compact.*

The first step to show existence of a thin flow with resetting is to get rid of Condition (9), which is done in the next lemma.

Lemma 1. *Let (G, u, s, t) be an acyclic network, $E_1 \subset E(G)$ and $F \geq 0$. Suppose that there exists an F-flow x' on G and a nonnegative node labeling ℓ' satisfying conditions (5) to (8). Then ℓ' can be modified in at most $O(|V(G)|^2)$ operations so that the new ℓ' is a thin flow of value F with resetting on E_1 and x' is an associated static flow.*

Proof. We only need to modify the labeling ℓ' so that (9) holds. Suppose that (9) fails at a certain $w \in V$ so that $\delta^-_G(w) \cap E_1 = \emptyset$ and $\ell'_w < \min_{vw \in \delta^-_G(w)} \ell'_v$. From (7) it

follows that for every edge $e = vw \in \delta_G^-(w)$ we must have $x'_e = 0$. Hence, redefining $\ell'_w := \min_{vw \in \delta_G^-(w)} \ell'_v$, the modified labels ℓ' still satisfy (5)-(8) while (9) now holds at an additional node w. Repeating this procedure with all the nodes w that violate (9), we get an ℓ' satisfying (5)-(9) in $O(|V(G)|^2)$. □

The following lemma relates a thin flow with resetting to the solutions of an appropriate nonlinear complementarity problem.

Lemma 2. *Let (G, u, s, t) be a network, $E_1 \subset E(G)$ and $F \geq 0$. Consider the mapping $\Phi : \mathbb{R}_+^{E(G)\cup V(G)} \to \mathbb{R}^{E(G)\cup V(G)}$ defined as*

$$\Phi_i(x', \ell') := \begin{cases} \max\{\ell'_v, x'_e/u_e\} - \ell'_w & \text{if } i = e = vw \in E(G)\setminus E_1 \\ x'_e/u_e - \ell'_w & \text{if } i = e = vw \in E_1 \\ \ell'_s - 1 & \text{if } i = s \\ \sum_{e\in\delta^-(v)} x'_e - \sum_{e\in\delta^+(v)} x'_e & \text{if } i = v \in V(G)\setminus\{s,t\} \\ \sum_{e\in\delta^-(t)} x'_e - \sum_{e\in\delta^+(t)} x'_e - F & \text{if } i = t\,. \end{cases} \tag{10}$$

Then, every solution (x', ℓ') of $\mathrm{NCP}(\Phi)$ satisfies conditions (5) to (8) and induces a feasible static flow of value F. Reciprocally, every thin flow of value F with resetting on E_1 is a solution of $\mathrm{NCP}(\Phi)$.

Proof. Let (x', ℓ') be a solution of $\mathrm{NCP}(\Phi)$. Let us check that conditions (5)-(8), and the flow constraints are verified. For this we study each of the following cases:

(i) $e \in E(G)\setminus E_1$. If $x'_e > 0$ then $\ell'_w = \max\{\ell'_v, x'_e/u_e\}$. Also, if $x'_e = 0$, since $\max\{\ell'_v, x'_e/u_e\} - \ell'_w \geq 0$ then $\ell'_w \leq \max\{\ell'_v, x'_e/u_e\} = \ell'_v$. Thus, conditions (6) and (7) are satisfied.

(ii) $e \in E_1$. If $x'_e > 0$ then $\ell'_w = x'_e/u_e$, while if $x'_e = 0$, since $x'_e/u_e - \ell'_w \geq 0$ and $\ell'_w \geq 0$ then $\ell'_w = x'_e/u_e = 0$. Thus, (8) is satisfied.

(iii) If $\ell'_s > 0$ then $\ell'_s = 1$. The case $\ell'_s = 0$ is not possible because $\ell'_s - 1 \geq 0$. Thus, (5) is satisfied.

(iv) $v \in V(G)\setminus\{s,t\}$. Assume $\ell'_v = 0$ and $\sum_{e\in\delta^-(v)} x'_e - \sum_{e\in\delta^+(v)} x'_e > 0$, then there exists $e \in \delta^-(v)$ such that $x'_e > 0$, which implies (by (7) or (8)) that $\ell'_v > 0$, a contradiction. If $\ell'_v > 0$ then $\sum_{e\in\delta^-(v)} x'_e - \sum_{e\in\delta^+(v)} x'_e = 0$. Thus, $\sum_{e\in\delta^-(v)} x'_e - \sum_{e\in\delta^+(v)} x'_e = 0$ for all $v \in V(G)\setminus\{s,t\}$. Similarly, for t we have $\left(\sum_{e\in\delta^-(t)} x'_e - \sum_{e\in\delta^+(t)} x'_e\right) - F = 0$. We conclude that $(x'_e)_{e\in E(G)}$ is a conservative F-flow.

The reciprocal assertion is verified directly. □

Theorem 4. *Let (G, u, s, t) an acyclic network, $E_1 \subset E(G)$ and $F \geq 0$. Then, there exists a thin flow of value F with resetting on E_1.*

Proof. Let Φ as in (10). By Lemmas 1 and 2, and since $\mathrm{VI}(\mathbb{R}_+^{E(G)\cup V(G)}, \Phi)$ is equivalent to $\mathrm{NCP}(\Phi)$, we need to prove that $\mathrm{VI}(\mathbb{R}_+^{E(G)\cup V(G)}, \Phi)$ has a solution.

Let δ and ϵ positive numbers such that $\epsilon > \delta$. Define $u_{min} := \min_{e\in E(G)} u_e$, $M := \max\{1, F/u_{min}\}$ and $K \subset \mathbb{R}_+^{E(G)\cup V(G)}$ as the compact convex set

$$K := \Big\{(x', \ell') \in \mathbb{R}_+^{E(G)\cup V(G)} : \forall e \in E(G), 0 \le x'_e \le (M+\epsilon)u_e, \\ \forall v \in V(G), 0 \le \ell'_v \le M + \delta\Big\}.$$

By Theorem 3 there exist a solution $(x', \ell') \in K$ of VI(K, Φ). Unfortunately, this solution may not coincide with a solution to NCP(Φ) since it can lie in the frontier of K. However, if there is an interior solution in K of VI(K, Φ), it is easily proven that such solution also solves VI$(\mathbb{R}_+^{E(G)\cup V(G)}, \Phi)$. The rest of the proof is technical and is devoted to prove that there is an interior solution of VI(K, Φ), i.e., a solution satisfying:

$$\forall e \in E(G), x'_e < (M+\epsilon)u_e \quad \text{and} \quad \forall v \in V(G), \ell'_v < M + \delta. \tag{P}$$

Suppose for a contradiction that there exists $e = vw \in E(G)$ such that $x'_e = (M+\epsilon)u_e$. Define $(y, h) \in K$ such that $y_e := 0$, $y_a := x'_a$ for every $a \in E(G) \setminus \{e\}$ and $h := \ell'$. Because (x', ℓ') is solution of VI(K, Φ) we have that

$$-(M+\epsilon)u_e\,(M + \epsilon - \ell'_w) \ge 0.$$

Therefore $(M + \epsilon - \ell'_w) \le 0$, so that $M + \epsilon \le \ell'_w \le M + \delta < M + \epsilon$, a contradiction. Thus, we have that for every $e \in E(G), x'_e < (M+\epsilon)u_e$,which together with the fact that (x', ℓ') solves VI(K, Φ), implies the complementarity condition

$$x'_e[\Phi(x', \ell')]_e = 0 \quad \text{for every } e = vw \in E(G). \tag{CC}$$

Therefore for every $e = vw \in E_1$, $x'_e = \ell'_w u_e$. Indeed, if $x'_e > 0$ this follows directly form (CC). Otherwise, if $x'_e = 0$, this follows by taking an element in K which equals (x', ℓ') in every coordinate except in $e = vw$, where it takes some positive value. The variational inequality then says that $\ell'_w \le x'_e/u_e = 0$.

Note that $\ell'_s = 1$. Define $b(v) = 0$ for every $v \in V \setminus \{s, t\}$ and $b(t) = -F$. Then, if $(y, h) \in K$ we have,

$$\begin{aligned}
&\left(\begin{pmatrix} y \\ h \end{pmatrix} - \begin{pmatrix} x' \\ \ell' \end{pmatrix}\right)^t \Phi(x', \ell') \\
&= \sum_{e=vw\in E\setminus E_1} (y_e - x'_e)(\max\{\ell'_v, x'_e/u_e\} - \ell'_w) + \sum_{e=vw\in E_1} (y_e - x'_e)(x'_e/u_e - \ell'_w) \\
&\quad + \sum_{v\in V\setminus\{s\}} (h_v - \ell'_v)\Big(\sum_{e\in\delta^-(v)} x'_e - \sum_{e\in\delta^+(v)} x'_e + b(v)\Big) \\
&= \sum_{e=vw\in E\setminus E_1 : x'_e = 0} y_e(\ell'_v - \ell'_w) + \sum_{v\in V\setminus\{s\}} (h_v - \ell'_v)\Big(\sum_{e\in\delta^-(v)} x'_e - \sum_{e\in\delta^+(v)} x'_e + b(v)\Big),
\end{aligned}$$

where the last equality follows from (CC).

Now we show that for every $v \in V(G)$, $\ell'_v < M + \delta$. Since G is acyclic, consider a topological ordering $v_1, v_2, ..., v_{|V(G)|}$ of $V(G)$ and let $w := v_{|V(G)|}$. By contradiction,

suppose that $\ell'_w = M + \delta$, and define the minimal $k \in \{1, ..., |V(G)|\}$ such that every node label in $T := \{v_k, ..., v_{|V(G)|}\}$ equals ℓ'_w. Note that $\ell'_s = 1$, then $T \neq V(G)$, then there exists $a = uv \in \delta^-(T)$ such that $\ell'_u < \ell'_v$. Observe that if $a \notin E_1$ then $x'_a > 0$. Indeed, suppose that $x'_a = 0$ and define $(y, h) \in K$ such that $h_v := \ell'_v$ for every $v \in V(G)$, $y_e = x'_e$ if $e \neq a$, and $y_a > 0$. Then, because $\ell'_u < \ell'_w$, we have that

$$\left(\begin{pmatrix} y \\ h \end{pmatrix} - \begin{pmatrix} x' \\ \ell' \end{pmatrix}\right)^t \Phi(x', \ell') = y_a(\ell'_u - \ell'_w) < 0,$$

a contradiction. Thus, $x'_a = \ell'_v u_a$, since if $a \notin E_1$ we have just shown that $x'_a > 0$ so that from (CC) $\ell'_v = \max\{\ell'_u, x'_a/u_a\} = x'_a/u_a$, and if $a \in E_1$ this was shown above.

Define $(y, h) \in K$ such that $h_v := \ell'_v$ for every $v \in V(G) \setminus T$, $h_v := 0$ for every $v \in T$ and $y = x'$. Then, we obtain a contradiction since

$$\begin{aligned}\left(\begin{pmatrix} y \\ h \end{pmatrix} - \begin{pmatrix} x' \\ \ell' \end{pmatrix}\right)^t \Phi(x', \ell') &= -\ell'_w\Big(\sum_{e\in\delta^-(T)} x'_e - \sum_{e\in\delta^+(T)} x'_e + \sum_{v\in T} b(v)\Big) \\ &= -\ell'_w\Big(\sum_{e\in\delta^-(T)} x'_e + \sum_{v\in T} b(v)\Big) = -\ell'_w(\ell'_w \sum_{e\in\delta^-(T)} u_e - F) \\ < -(\ell'_w \sum_{e\in\delta^-(T)} u_e - F) &< -((\max\{1, F/u_{min}\}u_a + \delta u_a) - F) < 0.\end{aligned}$$

Then, $\ell'_v < M + \delta$ for every $v \in T$, which implies the complementarity condition

$$\ell'_v[\Phi(x', \ell')]_v = 0, \qquad \text{for every } v \in T.$$

Therefore, if $\ell'_w > 0$, for every $v \in T$ we have $\sum_{e\in\delta^-(v)} x'_e - \sum_{e\in\delta^+(v)} x'_e + b(v) = 0$. In other words either x' satisfies flow conservation in every node in T or $\ell'_w = 0$ (and also $\ell'_v = 0$ for all $v \in T$), and in the latter case $x'(\delta^-(T)) = 0$.

If $T \cup \{s\} \neq V(G)$, we consider the graph G' obtained by deleting the node set T and the edge set $\cup_{v\in T}\delta^-(v) \cup \delta^+(v)$, and redefine $b(v) := b(v) - x'(\delta^+_G(v) \cap \delta^-(T))$. Thus $\sum_{v\in V(G)} b(v) = -F$, so that we can repeat the argument until $V(G') = \{s\}$. Because in each repetition at least one node is removed, this procedure finishes, concluding that $\ell'_v < M + \delta$ for every $v \in V(G)$. □

3.2 Uniqueness of Thin Flows with Resetting

We now show that thin flows with resetting are unique on acyclic networks. This means that the labeling $(\ell'_v)_{v\in V}$ is unique, but not necessarily the static flow $(x'_e)_{e\in E}$.

Theorem 5. *Let (G, u, s, t) be an acyclic network, $E_1 \subset E(G)$ and $F \geq 0$. Then, there exists a unique thin flow of value F with resetting on E_1.*

Proof. Let ℓ' and h' two thin flows of value F with resetting on E_1, and let x' and y' be corresponding static flows. We define the node set

$$S := \{v \in V(G) \mid \ell'_v \geq h'_v\}.$$

We first show the following property:

$$e = vw \in \delta_G^-(S) \Longrightarrow x'_e \geq y'_e. \tag{P_1}$$

In order to prove (P_1), given an edge $e = vw \in \delta_G^-(S)$, we consider three cases:

- Case $e \in E_1$: in this case $x'_e = \ell'_w u_e \geq h'_w u_e = y'_e$.
- Case $e \notin E_1$ and $\ell'_v \geq \ell'_w$: since $v \notin S$ we have $h'_v > \ell'_v \geq \ell'_w \geq h'_w$, then $h'_v > h'_w$ and therefore $y'_e = 0$, which implies $x'_e \geq y'_e$.
- Case $e \notin E_1$ and $\ell'_v < \ell'_w$: in this case $x'_e = \ell'_w u_e \geq h'_w u_e \geq y'_e$.

Similarly, we can show the following:

$$e = vw \in \delta_G^+(S) \Longrightarrow y'_e \geq x'_e. \tag{P_2}$$

Indeed, given an edge $e = vw \in \delta_G^+(S)$, we consider three cases:

- Case 1, $e \in E_1$: Since $w \notin S$ then $h'_w > \ell'_w$. Since $e \in E_1$, we have $x'_e = \ell'_w u_e$ and $y'_e = h'_w u_e$, thus $y'_e > x'_e$.
- Case 2, $e \notin E_1$ and $h'_v \geq h'_w$: Since $w \notin S$ then $h'_w > \ell'_w$. Since $v \in S$, we have $\ell'_v \geq h'_v$. Thus $\ell'_v > \ell'_w$ and therefore $x'_e = 0$, which implies $y'_e \geq x'_e$.
- Case 3, $e \notin E_1$ and $h'_v < h'_w$: Clearly $y'_e = h'_w u_e > \ell'_w u_e \geq x'_e$. Thus, $y'_e > x'_e$.

Consider $x'(\delta_G(S)) := \sum_{e \in \delta_G^+(S)} x'_e - \sum_{e \in \delta_G^-(S)} x'_e$. It follows from (P_1) and (P_2) that $x'(\delta_G(S)) \leq y'(\delta_G(S))$. On the other hand, we know that $x'(\delta_G(S)) = y'(\delta_G(S))$ because both flows are conservative. Then, for every $e \in \delta_G^+(S)$, Case 1 and Case 3 (where $y'_e > x'_e$) cannot hold, and therefore the only possible scenario is Case 2, so that

$$\text{for all } e \in \delta_G^+(S),\ x'_e = 0 \text{ and } e \notin E_1. \tag{P_3}$$

To finish the proof we study two cases, depending whether $F > 0$ or $F = 0$.
Case $F > 0$: From (P_3) we have

$$x'(\delta_G(S)) = \sum_{e \in \delta_G^+(S)} x'_e - \sum_{e \in \delta_G^-(S)} x'_e = - \sum_{e \in \delta_G^-(S)} x'_e. \tag{11}$$

On the other hand, we know that $s \in S$. This implies that if $t \notin S$, then (since $F > 0$) there exists $e \in \delta_G^+(S)$ such that $x'_e > 0$, contradicting P_3. Thus $s, t \in S$, and therefore $x'(\delta_G(S)) = 0$, which together with equation (11) implies that

$$\text{for all } e \in \delta_G^-(S),\ x'_e = 0. \tag{P_4}$$

From properties (P_1) and (P_4) we conclude that, for all $e \in \delta_G^-(S)$, $y'_e = 0$, and therefore, from the conservation constraints of the static flow y', we have that for all $e \in \delta_G^+(S)$, $y'_e = 0$. We have proved that

$$\text{for all } e \in \delta_G^+(S),\ x'_e = y'_e = 0 \text{ and } e \notin E_1. \tag{P_5}$$

Suppose there exists $w \in V(G) \setminus S$, i.e., $\ell'_w < h'_w$. Since G is acyclic, we can choose w such that $\delta_G^-(w) \subset \delta_G^+(S)$, and therefore, from (P_5), every $e \in \delta_G^-(w)$ satisfies

$x'_e = y'_e = 0$ and $e \notin E_1$. It follows from (9) that $\ell'_w = \min_{vw \in \delta^-_G(w)} \ell'_v$ and $h'_w = \min_{vw \in \delta^-_G(w)} h'_v$. However, since any $e = vw \in \delta^-_G(w)$ satisfies $v \in S$, we have

$$\ell'_w = \min_{vw \in \delta^-_G(w)} \ell'_v \geq \min_{vw \in \delta^-_G(w)} h'_v = h'_w,$$

contradicting that $w \in V(G) \setminus S$. We conclude that $S = V(G)$, i.e., for all $v \in V(G)$, $\ell'_v \geq h'_v$. Because the result is symmetric with respect to ℓ' and h', we have that $\ell' = h'$.
Case $F = 0$: We have that $e \in \delta^+_G(S)$, $x'_e = y'_e = 0$ and $e \notin E_1$. Repeating the argument of the case $F > 0$ yields the desired result. □

4 Existence and Uniqueness of Equilibria for Flows over Time

4.1 Existence for Piecewise Constant Inflow Rate

Koch and Skutella [9] showed a method to extend an equilibrium in case the network inflow rate function u_0 is constant, say $u_0 = F$. Given a feasible flow over time f which satisfies the equilibrium conditions in $[0, T]$ (i.e., flow is only sent along currently shortest paths), the method to extend this equilibrium is as follows:

1. Define the edge set $E_1 := \{e = vw \mid q_e(\ell_v(T)) > 0\} \subset E(G_T)$ and find a thin flow of value F with resetting on E_1 on the network G_T, say (ℓ', x').
2. Find the largest $\alpha > 0$ such that:

$$\ell_w(T) + \alpha\ell'_w - \ell_v(T) - \alpha\ell'_v \leq \tau_e \qquad \text{for all } e = vw \in E \setminus E(G_T), \tag{12}$$
$$\ell_w(T) + \alpha\ell'_w - \ell_v(T) - \alpha\ell'_v \geq \tau_e \qquad \text{for all } e = vw \in E_1. \tag{13}$$

3. Update the flow over time parameters in this order for all $\theta \in [T, T + \alpha]$

$$\begin{aligned} \ell_v(\theta) &:= \ell_v(T) + (\theta - T)\ell'_v && \text{for all } v \in V, \\ f^+_e(\ell_v(\theta)) &:= x'_e/\ell'_v && \text{for all } e = vw \in E(G_T), \\ f^-_e(\ell_w(\theta)) &:= x'_e/\ell'_w && \text{for all } e = vw \in E(G_T). \end{aligned}$$

From Theorem 4 we know that there exists a thin flow with resetting on Step 1, thus we can always extend an equilibrium. Because the label functions are non-decreasing and bounded in every set of the form $[0, T]$, that extension can be repeated as many times as wanted. Thus, starting from the interval $[0, 0]$, we can iterate this method to find a new $\alpha_i > 0$ and a new extension at every iteration i. Eventually, $\sum_{i=1}^n \alpha_i$ can have a limit point: in this case, since the label functions are non-decreasing and bounded in every set of the form $[0, T]$, we can define the equilibrium at point $\sum_{i=1}^\infty \alpha_i$ as the limit point of the label functions ℓ, and continue with the extension method. Note that, with Condition (9) at hand, this extension method works even if the inflow rate function is piecewise constant (by Theorem 2). So we have the following existence result.

Theorem 6. *Suppose that u_0 is piecewise constant, i.e, there exists a sequence $(\xi_k)_{k\in\mathbb{N}}$ such that $\xi_0 = 0$, $\sum_k \xi_k = \infty$ and u_0 restricted to the interval $[\xi_k, \xi_{k+1})$ is constant. Then, there exists a flow over time f which is an equilibrium whose network inflow rate is u_0 and whose label functions ℓ are piecewise linear.*

4.2 Uniqueness for Piecewise Constant Inflow Rate

Theorems 2 and 5, imply that there is a unique equilibria for flows over time (in the sense of labels) that can be constructed by the previous extension method. However, other type of bizarrely behaving equilibria might exist. To rule out this possibility it would be enough to show that at every iteration of the extension method $\alpha_i > \varepsilon > 0$.

Theorem 7. *Suppose that u_0 is piecewise constant. Then, there exists a unique array of label function $(\ell_v)_{v\in V}$ for every equilibrium in the family of flows over time which satisfy that for every $v \in V$ and every $\theta \in \mathbb{R}_+$, there exists $\epsilon > 0$ such that ℓ_v restricted to $[\theta, \theta + \epsilon]$ is an affine function.*

4.3 Existence for General Inflow Rate

In the previous sections, we showed existence in the case of the network inflow rate function u_0 is piecewise constant. We can show existence of equilibria for much more general inflow rate functions, namely when $u_0 \in L^p(\mathbb{R}_+)$, with $1 < p < \infty$ (i.e., u_0 is measurable and $\int_0^\infty |u_0(t)|^p dt < \infty$, see e.g. [3]). The proof, found in the full version of the paper, is neither algorithmic nor constructive, it is based on functional analysis techniques and on finding solutions to an appropriate variational problem.

Theorem 8. *Let p be such that $1 < p < \infty$. Let $u_0 \in L^p(\mathbb{R}_+)$ be a network inflow rate function of the network (G, u, τ, s, t). Then, there is an equilibrium s-t flow over time.*

Acknowledgements. We thank Martin Skutella for many stimulating discussions.

References

1. Anderson, E.J., Philpott, A.B.: Optimisation of flows in networks over time. In: Kelly, F.P. (ed.) Probability, Statistics and Optimisation, pp. 369–382. Wiley, New York (1994)
2. Bhaskar, U., Fleischer, L., Anshelevich, E.: A stackelberg strategy for routing flow over time. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 192–201 (2011)
3. Brezis, H.: Functional Analysis, Sobolev Spaces and Partial Differential Equations. Springer, New York (2010)
4. Facchinei, F., Pang, J.: Finite-Dimensional Variational Inequalities and Complementarity Problems, vol. I. Springer, New York (2003)
5. Fleischer, L., Tardos, E.: Efficient continuous-time dynamic network flow algorithms. Operations Research 23(3-5), 71–80 (1998)
6. Ford, L.R., Fulkerson, D.R.: Constructing maximal dynamic flows from static flows. Operations Research 6, 419–433 (1958)
7. Ford, L.R., Fulkerson, D.R.: Flows in Networks. Princeton University Press, Princeton (1962)
8. Gale, D.: Transient flows in networks. Michigan Mathematical Journal 6, 59–63 (1959)
9. Koch, R., Skutella, M.: Nash equilibria and the price of anarchy for flows over time. Theory of Computing Systems (to appear)
10. Macko, M., Larson, K., Steskal, Ľ.: Braess's Paradox for Flows over Time. In: Kontogiannis, S., Koutsoupias, E., Spirakis, P.G. (eds.) Algorithmic Game Theory. LNCS, vol. 6386, pp. 262–275. Springer, Heidelberg (2010)
11. Ran, B., Boyce, D.: Modeling Dynamic Transportation Networks. Springer, Berlin (1996)
12. Skutella, M.: An introduction to network flows over time. In: Cook, W., Lovasz, L., Vygen, J. (eds.) Research Trends in Combinatorial Optimization, pp. 451–482. Springer, Berlin (2009)

Collusion in Atomic Splittable Routing Games*

Chien-Chung Huang

Humboldt-Universität zu Berlin, Unter den Linden 6,
10099, Berlin, Germany
villars@informatik.hu-berlin.de

Abstract. We investigate how collusion affects the social cost in atomic splittable routing games. Suppose that players form coalitions and each coalition behaves as if it were a single player controlling all the flows of its participants. It may be tempting to conjecture that the social cost would be lower after collusion, since there would be more coordination among the players. We construct examples to show that this conjecture is not true. These examples motivates the question: *under what conditions would the social cost of the post-collusion equilibrium be bounded by the social cost of the pre-collusion equilibrium?*

We show that if (i) the network is "well-designed" (satisfying a natural condition), and (ii) the delay functions are affine, then collusion is always beneficial for the social cost in the Nash equilibria. On the other hand, if either of the above conditions is unsatisfied, collusion can worsen the social cost. Our main technique is a novel flow-augmenting algorithm to build Nash equilibria.

1 Introduction

In an *atomic splittable routing game*, each player controls a non-negligible amount of flow and he can route his flow fractionally over the network. His strategy space consists of all possible ways of routing his flow. Each edge of the network is associated with a *delay function* of the flow value. A player routing his flow on an edge incurs a *cost*, which is the product of his flow value on that edge and the delay of that edge, determined by the total flow value on that edge. A player's total cost is the sum of his cost on all edges. Players are selfish. They aim for minimizing their own total costs while disregarding the cost of others. The *social cost* is the sum of all players' costs.

Atomic splittable routing games abstract real world situations such as communication networks and transportation, where a player can be an ISP or a freight/logistic company and he would try to minimize the delay experienced by his customers. A special case of this setting, mentioned as early as 1952 by Wardrop [15], is where each player controls an infinitesimal amount of flow. In the scenarios above, these players could be individual messages or drivers. A player controlling an infinitesimal amount of flow is conventionally called a *nonatomic* player. Furthermore, a Nash equilibrium in which all players are nonatomic is often called a *Wardrop equilibrium*.

* Research supported by an Alexander von Humboldt fellowship.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 564–575, 2011.

We are interested in the social cost of a Nash equilibrium, a situation where no player can change his strategy unilaterally to decrease his cost. It can be expected that, given the selfishness of the players, the social cost of a Nash equilibrium would be sub-optimal. The degree of the worsening of the social cost in Nash equilibria is measured by the *price of anarchy* [10], and it has been intensively studied in recent years [5, 7, 11–13].

Life can be a bit more complex. Players form coalitions; companies merge or cooperate. So we are concerned about the social cost of the Nash equilibrium after the collusion. There can be different models to describe the colluding behavior among the players. The one we adopt is introduced by Hayrapetyan, Tardos, and Wexler [8]. In this model, once a coalition is formed, its participants care about their collective welfare. Thus this coalition would behave as if it were just a single player controlling all the flows of its participants.

It may be tempting to conjecture that the social cost will decrease after the collusion, since the colluding players have better coordination among themselves. In the extreme case where all players join in a single coalition, then the resultant equilibrium would become optimal. However, this conjecture has been shown to be false in several types of games [8].

For atomic splittable flow games, in this paper, we construct examples to show that even in very simple single-source-single-destination networks, collusion can worsen the social cost in the post-collusion equilibrium. These examples prompt us to investigate the following question:

> In an atomic splittable routing game, suppose that all players share a common source and destination. Under what conditions would the social cost of the post-collusion equilibrium be bounded by the social cost of the pre-collusion equilibrium?

Our Contribution

We first introduce a class of networks, which is a main focus of this work. Let the optimal flow be the flow that minimizes the social cost.

Definition 1. *A single-source-single-destination network is* well-designed*, if as the value of the optimal flow is being increased, its flow value is monotonically non-decreasing on all edges. Precisely, let $O(t)$ denote an optimal flow of value t, indexed over all edges $e \in E$. A single-source-single-destination network is* well-designed*, if $t > t'$, then $O_e(t) \geq O_e(t')$ for all edges $e \in E$.*

A well-designed network thus conforms to the intuition that as the total flow becomes larger, we expect each edge to be more heavily used. In general, whether a network is well-designed depends on both the underlying graph topology and the delay functions. But there is an exception.

Proposition 1. *Suppose that the underlying graph of the network is series-parallel and all delay functions are convex. Then such a network is always well-designed, independent of the convex delay functions.*

The proof of this proposition and all other missing proofs can be found in the full version [9].

We now state our main result.

Theorem 1. *Suppose that the given network has a single source and a single destination and all delay functions are convex. If*

(i) the network is well-designed, and
(ii) all delay functions are affine,

then the social cost of the post-collusion equilibrium is bounded by that of the pre-collusion equilibrium.

On the other hand, if either of the two conditions is unsatisfied, then the social cost of the post-collusion equilibrium can be strictly larger than that of the pre-collusion equilibrium.

Our Technique. Let $\sigma = (v_1, v_2, \cdots, v_k)$ be a *profile*, where $v_1 \geq v_2 \geq \cdots \geq v_k$ and each v_i represents the flow value of the i-th player. Note that if there are $k' < k$ players, we still can regard the game as one of k players with the last $k - k'$ players having zero flow values, i.e., the last $k - k'$ entries of σ are 0.

Definition 2. *Let $\sigma = (v_1, v_2, \cdots, v_k)$ and $\sigma' = (v'_1, v'_2, \cdots, v'_k)$ be two profiles and $\sum_{i=1}^{k} v_i = \sum_{i=1}^{k} v'_i = 1$. σ* majorizes *σ' if, for all $1 \leq i \leq k$, $\sum_{j=1}^{i} v_j \geq \sum_{j=1}^{i} v'_j$.*

We establish the first part of Theorem 1 via the following lemma.

Lemma 1. *Let σ and σ' be two profiles where the former majorizes the latter. In a well-designed network with affine delays, the social cost of the Nash equilibrium for σ is bounded by the social cost of the Nash equilibrium for σ'.*

Proof of the First Part of Theorem 1. Let σ' be the profile of the given game, and let σ be the profile after some players form coalitions. Observe that σ must majorize σ'. Hence Lemma 1 gives the proof. □

The main bulk of this paper (Section 3) is devoted to proving Lemma 1. Here we present the high-level idea.

We first characterize a well-designed network and show that the optimal flow is updated according to certain rate equations when its flow value is being increased in such a network. Exploiting these rate equations, we design a flow-augmenting algorithm to build a Nash equilibrium. Then we apply this algorithm to the input profiles σ and σ' simultaneously. We show that the derivative of the social cost for the flow based on σ is always no larger than that for the flow based on σ', thereby proving Lemma 1.

Counter-Examples. Our positive result for collusion is tight: dropping either of the two conditions may cause collusion to become detrimental for the social cost. We carefully construct two counter-examples in which after some players

form coalitions, the social cost in the post-collusion equilibrium is strictly higher than in the pre-collusion equilibrium. We briefly summarize the two examples (see the full version for details):

- *When the network has affine delays but is not well-designed*: in the Braess graph, which is known to be the smallest non-series-parallel graph, we build a not well-designed network with affine delays.
- *When the network is well-designed but the delay functions are not affine*: we use a network of just 3 parallel links (a special case of a series-parallel graph, hence well-designed). This network has polynomials as delay functions.

Computational Result. Given Theorem 1, it would be of practical interest to test whether a network is well-designed.

Theorem 2. *Suppose that all delay functions are affine and the coefficients of the delays are rational. There is a polynomial time algorithm to test whether a network is well-designed. Moreover, if it is, we can find the* exact *Nash equilibrium and the* exact *Wardrop equilibrium in polynomial time.*

When all delay functions are affine, it is known that one can use convex programming to approximate the Nash equilibrium [5] or to approximate the Wardrop equilibrium [14]. To our knowledge, our algorithm is the first to find the exact equilibria for a nontrivial sub-class of atomic splittable routing games. See the full version for details about how to test a network is well-designed.

Related Work. Atomic splittable routing games are the least understood among various versions of selfish routing games. The exact price of anarchy in such games was only recently obtained by Roughgarden and Schoppmann [13].

Hayrapetyan, Tardos, and Wexler [8] investigated the effect of collusion in various games by measuring the *price of collusion*, which is the worst ratio between the social cost of a post-collusion equilibrium against that of a pre-collusion equilibrium. Using their terminology, our results can be rephrased as identifying the conditions in atomic splittable routing games for the price of collusion to be bounded by 1. Fotakis, Kontogiannis, and Spirakis [6] investigated algorithmic questions about the Nash equilibrium after collusion in atomic congestion games.

When collusion is among non-atomic players. A closely related scenario about collusion is that initially each player is nonatomic, i.e., he controls an infinitesimal amount of flow. The players may form themselves into coalitions and each coalition cares about its collective welfare. Thus, each coalition would behave as if it were an atomic player controlling a splittable flow. In this scenario, the comparison between the social costs of the post-collusion equilibrium and of the pre-collusion *Wardrop* equilibrium is studied in [4, 5, 8]. In particular, these works investigate under what conditions would the social cost of the pre-collusion Wardrop equilibrium (with non-atomic players) be bounded by that of the post-collusion Nash equilibrium (with atomic players). Our work is the first to study the effect of collusion among the atomic players themselves.

In [8], it is shown that in a parallel-link graph with convex delays, the post-collusion Nash equilibrium has its social cost bounded by the social cost of the pre-collusion Wardrop equilibrium. The same result has been generalized to series-parallel graphs with convex delays in [4]. Our second counter-example offers an interesting contrast: even in a graph of just 3 links, if the collusion is among the atomic players themselves, collusion can worsen the social cost.

In the case that the network is well-designed with affine delays, [4] shows that the post-collusion Nash equilibrium has its social cost bounded by the social cost of the pre-collusion Wardrop equilibrium. Lemma 1 can establish the same fact.

Other models of collusion. Another line of investigation about collusion is the *strong price of anarchy* [2] introduced by Andelman, Feldman, and Mansour, which is the worst ratio of the social cost in a *strong Nash equilibrium* [3] against that of the optimal solution. In a strong Nash equilibrium, no coalition of players can change their strategies so that every one of them improves his cost.

2 Preliminaries

Let $G = (V, E)$ be a directed graph, with two special vertices s and d called *source* and *destination* respectively. The vector f, indexed by edges $e \in E$, is defined as a *flow* of *value* v if the following conditions are satisfied.

$$\sum_{w:(u,w)\in E} f_{uw} - \sum_{w:(w,u)\in E} f_{wu} = 0, \quad \forall u \in V \backslash \{s, d\}. \tag{1}$$

$$\sum_{w:(s,w)\in E} f_{sw} - \sum_{w:(w,s)\in E} f_{ws} = v. \tag{2}$$

$$f_e \geq 0, \quad \forall e \in E. \tag{3}$$

A flow is a *circulation* if its value is 0. If f satisfies only conditions (1) and (2), f is a *pseudo flow* of value v. If there are several flows $\{f^1, f^2, \cdots, f^k\}$, the total flow f is defined as $f := \sum_{i=1}^{k} f^i$. We define $f^{-i} := \sum_{j\neq i} f^j = f - f^i$.

Each edge e is associated with a delay function $l_e : \mathcal{R}^+ \to \mathcal{R}^+$. A delay function is *affine*, if $l_e(x) = a_e x + b_e$, where $a_e > 0$ and $b_e \geq 0$. For a flow f, player i incurs a cost $f_e^i l_e(f_e)$ on edge e. His total cost is the sum of his cost over all edges $C^i(f) = \sum_{e\in E} f_e^i l_e(f_e)$. The total cost of a flow f is defined as $C(f) = \sum_{e\in E} f_e l_e(f_e)$, which is also the sum of the costs of all players $\sum_i C^i(f)$. A flow $O(t)$ of value t is optimal if it minimizes the social cost among all flows f of value t, that is $\sum_{e\in E} O_e(t) l_e(O_e(t)) \leq \sum_{e\in E} f_e l_e(f_e)$.

An *atomic splittable routing game* is a tuple $(\sigma, (G, \mathbf{l}, s, d))$ where $\sigma = (v_1, \cdots, v_k)$ is a profile indicating the flow values of the players from 1 to k, ordered non-increasingly, and $(G, \mathbf{l}, s, d)$ a network and $\mathbf{l}$ its vector of delay functions for edges in G. A s-d path is a directed simple path from s to d. $\mathcal{P}$ denotes the set of all s-d paths. We often abuse notation by writing $e \in \mathcal{P}' \subseteq \mathcal{P}$ if e belongs to any path $p \in \mathcal{P}'$. An edge e is *used* if $f_e > 0$ and is *used by player* i if $f_e^i > 0$. Similarly, a path p is used (by player i) if on every edge $e \in p$, $f_e > 0$ ($f_e^i > 0$).

Each player i has a strategy space consisting of all possible s-d flows of value v_i. His objective is to minimize his delay $C^i(f)$. A set of players are said to be *symmetric* if each of them has the same flow value.

A flow is defined as a Nash equilibrium if no player can unilaterally alter his flow to reduce his cost.

Definition 3 (Nash Equilibrium). *In an atomic splittable flow game $(\sigma, (G, \mathbf{l}, s, d))$, flow f is a Nash equilibrium if and only if, for every player i and every s-d flow g of value v_i, $C^i(f^i, f^{-i}) \leq C^i(g, f^{-i})$.*

For a player i and a path p, his *marginal cost on path p* is the rate of change of his cost when he adds flow along path p: $\sum_{e\in p} l_e(f_e) + f_e^i l'_e(f_e)$. The following lemma follows from Karush-Kuhn-Tucker optimality conditions for convex programs applied to player i's minimization problem.

Lemma 2. *Flow f is a Nash equilibrium if and only if for any player i and any two directed paths p and q between the same pair of vertices such that player i uses path p,*

$$\sum_{e\in p} l_e(f_e) + f_e^i l'_e(f_e) \leq \sum_{e\in q} l_e(f_e) + f_e^i l'_e(f_e).$$

Note that the Nash equilibrium of just one player is exactly the optimal flow.

Lemma 3. [1, 11] *In an atomic splittable routing game $(\sigma, (G, \mathbf{l}, s, d))$, if all functions in $\mathbf{l}$ are affine, or if the underlying graph of G is parallel-link and all functions in $\mathbf{l}$ are convex, then there exists a Nash equilibrium and it is unique.*

When All Delay Functions are Affine. We introduce the notion of *ϕ-delay*, which plays a prominent role in our algorithms and analysis. Let the *ϕ-delay* of an edge e be $L_e(f, \phi) = \phi a_e f_e + b_e$ and along a path p be $L_p(f, \phi) = \sum_{e\in p} L_e(f, \phi)$.

Definition 4. *A flow f is* ϕ-optimal *for some $\phi > 0$ if and only if, given any two directed paths p and q between the same pair of vertices and f uses path p,*

$$L_p(f, \phi) \leq L_q(f, \phi).$$

By Definition 4, a 2-optimal flow is exactly the optimal flow; 1-optimal flow is a Wardrop equilibrium[1]; and a Nash equilibrium of k symmetric players is $\frac{k+1}{k}$-optimal[2].

[1] A Wardrop equilibrium f has the following characterization [15]: if p and q are directed paths between the same pair of vertices and f uses p, $\sum_{e\in p} l_e(f_e) \leq \sum_{e\in q} l_e(f_e)$.

[2] This follows from the fact that in a Nash equilibrium of symmetric players, the flows of all players are identical [5].

3 Proof of Lemma 1

In this section, we assume that all delay functions are affine. We prove Lemma 1 through the following three steps: (1) we characterize well-designed networks (Section 3.1); (2) using this characterization, we design an algorithm to construct a Nash equilibrium based on the given profile (Section 3.2); (3) we apply this algorithm to two different profiles and observe their relative growing speeds of the social cost while the flow values are being increased (Section 3.3).

3.1 Characterizing Well-Designed Networks

A well-designed network is associated with several sets of items, whose exact properties will be captured by Lemma 4 below. Here we summarize them. A well-designed network $(G, \mathbf{l}, s, d)$ has

- a sequence of nested sets of paths $\mathcal{P}_0 = \emptyset \subset \mathcal{P}_1 \subset \mathcal{P}_2 \subset \cdots \mathcal{P}_x \subseteq \mathcal{P}$, which are the sets of s-d paths that the optimal flow uses when its flow value is increased;
- a sequence of flow values $t_0 = 0 < t_1 < \cdots < t_{x-1} < t_x = \infty$, which are the values by which the optimal flow begins to use a different set of paths[3];
- a sequence of vectors α^1, α^2, $\cdots$, α^x, which are vectors indexed over the edges in E. Each of these vectors indicates how the optimal flow updates itself when its flow value is increased;
- a sequence of *ϕ-delay* thresholds $\Psi_0 = \min_{q \in \mathcal{P}} \sum_{e \in q} b_e < \Psi_1 < \Psi_2 < \cdots < \Psi_{x-1}$, each of which indicates the 2-delay of a path in $\mathcal{P}_1$ when the flow value is t_i.

Before explaining the details about these items, we need a fact from linear algebra. Assume that $\mathcal{P}_i \subseteq \mathcal{P}$ is a subset of s-d paths used by a flow f of value t. Consider the following linear system.

$$\sum_{e \in p} a_e f_e - \sum_{e \in q} a_e f_e = \frac{1}{2}\left(\sum_{e \in q} b_e - \sum_{e \in p} b_e\right) \quad \forall p, q \in \mathcal{P}_i \tag{4}$$

$$\sum_{u:(s,u) \in E} f_{su} = t \tag{5}$$

$$\sum_{v:(u,v) \in E} f_{uv} - \sum_{v:(v,u) \in E} f_{vu} = 0 \qquad \forall u \notin \{s, d\} \tag{6}$$

$$f_e = 0 \qquad \forall e \notin \mathcal{P}_i \tag{7}$$

Proposition 2. *If the system (4)-(7) has a unique solution, then the flow value on each edge $e \in E$ can be expressed as $f_e = \alpha_e^i t + \beta_e^i$, where α_e^i and β_e^i depend on $\mathcal{P}_i$ and $\{a_e, b_e\}_{e \in \mathcal{P}_i}$.*

Lemma 4. *Suppose that $(G, \mathbf{l}, s, d)$ is well-designed.*

[3] To simplify our presentation, we slightly abuse notation by writing a value that goes to infinity as a fixed value t_x.

(i) There exists a sequence of nested sets of paths $\mathcal{P}_0 = \emptyset \subset \mathcal{P}_1 \subset \mathcal{P}_2 \subset \cdots \subset \mathcal{P}_x \subseteq \mathcal{P}$*, and a set of values* $t_0 = 0 < t_1 < \cdots < t_{x-1} < t_x = \infty$ *so that when* $t \in (t_{i-1}, t_i]$, $\forall 1 \le i \le x$, $O(t)$ *uses the set of edges in* $\mathcal{P}_i$.

(ii) There exists a sequence of vectors α^i, $1 \le i \le x$, *which has the following properties:*

(iia) if $t_{i-1} \le t < t' \le t_i$, *then* $O(t') - O(t) = \alpha^i(t' - t)$;

(iib) α^i, *by itself, is also a flow of value 1; specifically,* $\alpha^i_e \ge 0$, $\forall i, e$, *and* $\alpha^i_e = 0$ *if* $e \notin \mathcal{P}_i$;

(iic) For every two paths p,q $\in \mathcal{P}_i$, $\sum_{e\in p} a_e \alpha^i_e = \sum_{e\in q} a_e \alpha^i_e$.

(iii) There exists a sequence of ϕ-delay thresholds $\Psi_0 = \min_{q\in\mathcal{P}} \sum_{e\in q} b_e < \Psi_1 < \Psi_2 < \cdots < \Psi_{x-1}$ *so that when* $t = t_i$, $1 \le i \le x-1$, *all paths* $q \in \mathcal{P}_{i+1}$ *(including those in* $\mathcal{P}_{i+1} \backslash \mathcal{P}_i$*) have the same 2-delay* $\Psi_i = L_p(O(t_i), 2)$.

In the following, we assume that the nested sets of paths $\mathcal{P}_i$, the vectors α^i, and the ϕ-delay thresholds Ψ_i are known. See the full version for about finding them in polynomial time if all coefficients $\{a_e, b_e\}_{e\in E}$ are rational.

The Vectors α^i as "Accelerators" and ϕ-Delay Thresholds Ψ_i as "Landmarks". The vectors α^i and the ϕ-delay thresholds Ψ_i play a pivotal role in our algorithm in Section 3.2. Here we explain why they are useful.

Lemma 4(ii) suggests an algorithmic view on an optimal flow when its value is increased: when its value is increased from t to $t+\epsilon$ so that $t_{i-1} \le t \le t+\epsilon \le t_i$, the optimal flow is *increased in the speed of* α^i in the following sense

$$O(t+\epsilon) := O(t) + \alpha^i \epsilon.$$

In other words, the vector α^i serves as an accelerator to tell the optimal flow how it should update itself when the flow value increases. Lemma 4(iii) implies that once the flow value reaches t_i, the 2-delay of all paths in $\mathcal{P}_{i+1}$ will become exactly Ψ_i. And after this point, the optimal flow begins to use the paths in $\mathcal{P}_{i+1}$, instead of $\mathcal{P}_i$. This suggests that we can view these thresholds Ψ_i as a sort of "landmarks": pick any path $p \in \mathcal{P}_1$ (and note that then $p \in \mathcal{P}_i$ for any i). Once its 2-delay becomes Ψ_i, it is time to change gear: the optimal flow thence increases in the speed of α^{i+1}, instead of α^i.

Our main algorithm in Section 3.2 is inspired by these observations. Initially, the flow value is 0 and we increase it gradually up to 1. In this process, the flow is updated based on these accelerators α^i, and each time the ϕ-delay of a path in $\mathcal{P}_1$ reaches a landmark Ψ_i, we use a different accelerator α^{i+1} to update the flow.

To better illustrate our idea, we first present a simpler algorithm, which we call Algorithm A, in Figure 1. It constructs a Nash equilibrium for k symmetric players. Note that when $k \to \infty$, the Nash equilibrium would just be a Wardrop equilibrium.

Algorithm A maintains a $\frac{k+1}{k}$-optimal flow $f(t)$ of value t when t is increased from 0 to 1. The index $\mathtt{h}$ records the set of paths $\mathcal{P}_{\mathtt{h}}$ that $f(t)$ uses. $f(t)$ is increased in the speed of $\alpha^{\mathtt{h}}$ (see Lines 5-6). Each time the $\frac{k+1}{k}$-delay of a path $p \in \mathcal{P}_1$ reaches Ψ_h, $f(t)$ is then increased in the speed of α^{h+1} (see Lines 2 and 4). Line 3 records the value $t_h(k)$ by which $f(t)$ shifts from using $\mathcal{P}_h$ to $\mathcal{P}_{h+1}$.

Input: k // the number of symmetric players;
Initialization: $t := 0$; $f_e(t) := 0$, $\forall e \in E$; $\mathtt{h} = 1$; $\epsilon > 0$ is an infinitesimal amount;
// $f(t)$ is the current flow of value t; $\mathtt{h}$ is the index of the set of paths $\mathcal{P}_{\mathtt{h}}$ that $f(t)$ uses;
0. pick $p \in \mathcal{P}_1$;
1. **While** $t < 1$
2. **if** $L_p(f(t), \frac{k+1}{k}) = \Psi_h$ **then**
3. $t_{\mathtt{h}}(k) := t$;
4. $\mathtt{h} := \mathtt{h} + 1$;
5. $f(t + \epsilon) := f(t) + \alpha^h \epsilon$;
6. $t := t + \epsilon$;
7. **End**
8. $t_{\mathtt{h}}(k) := 1$;

Fig. 1. Algorithm A

Lemma 5. *Let $t_0(k) = 0$ and $t_z(k) = 1$. (So when Algorithm A terminates $\mathtt{h} = z$). Then for $0 \le i \le z$, $f(t_i(k))$ is an equilibrium flow with total value $t_i(k)$ for k symmetric players, and it uses the same set of paths $\mathcal{P}_i$ as $O(t_i)$.*

3.2 Constructing Nash Equilibria in Well-Designed Networks

The nice thing about a well-designed network is that it guarantees its Nash equilibria satisfy the *nesting property.*

Definition 5 (Nesting Property). *A flow f satisfies the* nesting property*, if when players i and j have flow values $v_i > v_j$, then $f_e^i > f_e^j$ on any edge $e \in E$ used by player j, and when $v_i = v_j$, then $f^i = f^j$.*

We give an alternative characterization of Nash equilibria if they satisfy the nesting property. Let

$$b_e^r := a_e\Big(\sum_{i=r+1}^{k} f_e^i\Big) + b_e.$$

Lemma 6. *Suppose that flow f satisfies the nesting property. If given any two directed paths p and q between the same pair of vertices and p is used by player r, and the following holds:*

$$\sum_{e \in p} a_e f_e^r + \frac{1}{r+1} b_e^r \le \sum_{e \in q} a_e f_e^r + \frac{1}{r+1} b_e^r,$$

then the flow f is a Nash equilibrium.

High-Level Ideas of Algorithm B. Let $\sigma = (v_1, \cdots, v_k)$ be the input profile. Our algorithm maintains a ϕ-optimal flow when the total flow value is increased from 0 to 1. But unlike the previous algorithm, ϕ is dynamically changing.

We maintain two indices $\mathtt{h}$ and $\mathtt{r}$, where $\mathtt{h}$ indicates the set of paths $\mathcal{P}_{\mathtt{h}}$ the current flow uses and $\mathtt{r}$ the index of the player whose flow is about to be created (we will explain how). Initially, let $\mathtt{h} := 1$ and $\mathtt{r} := k$.

While maintaining the invariant the current flow is $\frac{\mathtt{r}+1}{\mathtt{r}}$-optimal, our algorithm increases the flow in the speed of $\alpha^{\mathtt{h}}$. Two things may happen in this process.

- The $\frac{\mathtt{r}+1}{\mathtt{r}}$-delay of a path $p \in \mathcal{P}_1$ attains $\Psi_{\mathtt{h}}$. Then we increase the flow thence in the speed of $\alpha^{\mathtt{h}+1}$ (so we increment the index $\mathtt{h}$).
- The current flow value attains $\mathtt{r}v_{\mathtt{r}}$. Then we "freeze" $\frac{1}{\mathtt{r}}$ fraction of the current flow, give it to the $\mathtt{r}$-th player, treat the remaining unfrozen $\frac{\mathtt{r}-1}{\mathtt{r}}$ fraction as the current flow, and "update" the network (to be explained shortly).
  We will prove that the unfrozen $\frac{\mathtt{r}-1}{\mathtt{r}}$ fraction of the flow is $\frac{\mathtt{r}}{\mathtt{r}-1}$-optimal in the *updated* network. This is a crucial point of our algorithm. After the freezing, we decrement the index $\mathtt{r}$.

In the end, we can show that the k frozen flows that are assigned to the players satisfy the condition in (6) and hence by Lemma 6 they constitute a Nash equilibrium in the *original* network.

We now explain how the "freezing" is done and how the network is updated. Let $f(\sigma, t)$ denote the *current flow* and we will maintain the invariant that its flow value is $t - \sum_{j=\mathtt{r}+1}^{k} v_j$. Assume that in the current network the delay function of each edge e is $l_e(x) = a_e x + b_e^{\mathtt{r}}$ and the flow value of $f(\sigma, t)$ is $\mathtt{r}v_{\mathtt{r}}$. We freeze a fraction of the current flow as follows:

$$f^{\mathtt{r}} := \frac{1}{\mathtt{r}} f(\sigma, t), f(\sigma, t) := \frac{\mathtt{r}-1}{\mathtt{r}} f(\sigma, t).$$

Thus $f(\sigma, t)$ is split into two parts: the frozen part, $f^{\mathtt{r}}$, which is given to the $\mathtt{r}$-th player; the unfrozen part $\frac{\mathtt{r}-1}{\mathtt{r}} f(\sigma, t)$, which is now treated as the current flow of value $(\mathtt{r}-1)v_{\mathtt{r}}$.

We also update the network in the process of freezing as follows.

$$l_e(x) := a_e x + (b^{\mathtt{r}} + a_e f_e^{\mathtt{r}}) = a_e x + b_e^{\mathtt{r}-1}, \forall e \in E.$$

In words, the newly frozen flow $f^{\mathtt{r}}$ adds a constant term to the delay function on each edge e. Finally, we decrement the index $\mathtt{r}$.

Now let $L_{e,\mathtt{r}}(f(\sigma, t), \frac{\mathtt{r}+1}{\mathtt{r}})$ denote the $\frac{\mathtt{r}+1}{\mathtt{r}}$-delay of edge e in the current network (which has been updated $k - \mathtt{r}$ times because of those frozen flows), i.e.,

$$L_{e,\mathtt{r}}(f(\sigma, t), \frac{\mathtt{r}+1}{\mathtt{r}}) = \frac{\mathtt{r}+1}{\mathtt{r}} a_e f_e(\sigma, t) + b_e^{\mathtt{r}},$$

and in the following discussion we refer to $L_{e,\mathtt{r}}(f(\sigma, t), \frac{\mathtt{r}+1}{\mathtt{r}})$ as the ϕ-delay of the current flow on edge e.

We make a crucial observation about the consequence of the freezing: *The ϕ-delay of the current flow on each edge remains unchanged after the freezing.* To be precise, assume that after the freezing, the decremented index $\mathtt{r} = r - 1$.

Input: A profile $\sigma = (v_1, \cdots, v_k)$, where $\sum_{j=1}^{k} v_j = 1$;
Initialization: $t := 0$; $\mathtt{r} := k$; $\mathtt{h} := 1$; $f_e(\sigma, t) := 0, \forall e \in E$; $\epsilon > 0$ an infinitesimal amount;
// $f(\sigma, t)$ is the current flow, $\mathtt{h}$ is the index of the set of paths $\mathcal{P}_{\mathtt{h}}$ that $f(\sigma, t)$ uses;
// $\mathtt{r}$ is the index of the player whose flow is about to be created;
0. pick $p \in \mathcal{P}_1$;
1. **While** $t \leq 1$ **and** $\mathtt{r} \geq 1$
2. **if** $L_{p,\mathtt{r}}(f(\sigma, t), \frac{\mathtt{r}+1}{\mathtt{r}}) = \Psi_{\mathtt{h}}$ **then** // the ϕ-delay of p reaches a ϕ-delay threshold
3. $t_{\mathtt{h}}(\sigma) := t$;
4. $\mathtt{h} := \mathtt{h} + 1$;
5. **else if** $t - \sum_{j=\mathtt{r}+1}^{k} v_j = \mathtt{r}v_{\mathtt{r}}$ **then** // the current flow value reaches $\mathtt{r}v_{\mathtt{r}}$
6 $t^{\mathtt{r}}(\sigma) := t$;
7. $f^{\mathtt{r}} := \frac{1}{\mathtt{r}} f(\sigma, t)$;
8. $f(\sigma, t) := \frac{\mathtt{r}-1}{\mathtt{r}} f(\sigma, t)$;
9. $\forall e \in E$, $l_e(x) := a_e x + a_e f_e^{\mathtt{r}} + b_e^{\mathtt{r}} = a_e x + b_e^{\mathtt{r}-1}$;
10. $\mathtt{r} := \mathtt{r} - 1$;
11. **else**
12. $f(\sigma, t + \epsilon) := f(\sigma, t) + \alpha^{\mathtt{h}} \epsilon$;
13. $t := t + \epsilon$;
14. **End**
15. $t_{\mathtt{h}}(\sigma) := 1$;

Fig. 2. Algorithm B

Then *the* $\frac{r}{r-1}$*-delay of each edge (path) in the new network with the remaining unfrozen flow* $\frac{r-1}{r} f(\sigma, t)$ *would be identical to its* $\frac{r+1}{r}$*-delay in the previous network with its entire flow* $f(\sigma, t)$.

Claim. For each edge $e \in E$, $L_{e,r-1}(\frac{r-1}{r} f(\sigma, t), \frac{r}{r-1}) = L_{e,r}(f(\sigma, t), \frac{r+1}{r})$.

By this claim, if we can show that immediately before the freezing, the current flow is $\frac{r+1}{r}$-optimal in the previous network, then the unfrozen remaining flow will be $\frac{r}{r-1}$-optimal in the updated network.

We now present our main algorithm, Algorithm B, in Figure 2. We record the values $t_h(\sigma)$ and $t^r(\sigma)$ (see Lines 3 and 6) as the total flow values by which the current flow shifts to a larger set of paths and by which the flow of the r-th player is created.

Lemma 7. *f^i is a flow of value v_i for all $1 \leq i \leq k$. Furthermore, $f^1, \cdots, f^k$ satisfy the nesting property and they constitute a Nash equilibrium in the original network.*

3.3 Comparing the Social Cost of Two Different Profiles

We prove Lemma 1 by applying Algorithm B to two different profiles σ and σ' simultaneously.

Let $r(\sigma, t)$ denote the values of the indices $\mathtt{h}$ and $\mathtt{r}$ in the execution of Algorithm B for σ at time t. More precisely,

$$r(\sigma, t) = r \qquad \text{if } t \in (t^{r+1}(\sigma), t^r(\sigma)] \qquad (t^{k+1}(\sigma) \text{ is understood to be } 0)$$

Let $C(\sigma, t)$ be the social cost of the *accumulated flow* at time t, which is the sum of the current flow $f(\sigma, t)$ and those frozen flows, in the *original* network:

$$C(\sigma,t) = \sum_{e\in E}\left[f_e(\sigma,t) + \sum_{j=r(\sigma,t)+1}^{k} f_e^j\right]\left[a_e(f_e(\sigma,t) + \sum_{j=r(\sigma,t)+1}^{k} f_e^j) + b_e\right].$$

Proof of Lemma 1: We show that

$$\frac{dC(\sigma,t)}{dt}|_{t=t^*} \leq \frac{dC(\sigma',t)}{dt}|_{t=t^*} \quad \text{for all } 0 \leq t^* \leq 1,\ t^* \notin \{t_i(\sigma), t_i(\sigma')\}_{\forall i}.$$

Since $C(\sigma, t)$ and $C(\sigma', t)$ are both continuous, the comparison of their derivatives in the intervals suffices to establish Lemma 1. □

References

1. Altman, E., Basar, T., Jimenez, T., Shimkin, N.: Competitive routing in networks with polynomial costs. IEEE Transactions on Automatic Control 47(1), 92–96 (2002)
2. Andelman, N., Feldman, M., Mansour, Y.: Strong price of anarchy. In: SODA, pp. 189–198 (2007)
3. Aumann, R.: Acceptable points in geneeral cooperative n-person games. Contributions to the Theory of Games 4 (1959)
4. Bhaskar, U., Fleischer, L., Huang, C.-C.: The price of collusion in series-parallel networks. In: Eisenbrand, F., Shepherd, F.B. (eds.) IPCO 2010. LNCS, vol. 6080, pp. 313–326. Springer, Heidelberg (2010)
5. Cominetti, R., Correa, J., Stier-Moses, N.E.: The impact of oligopolistic competition in networks. Operations Research 57(6), 1421–1437 (2009)
6. Fotakis, D., Kontogiannis, S., Spirakis, P.: Atomic congestion games among coalitions. ACM Transactions on Algorithms 4(4) (2008)
7. Harks, T.: Stackelberg strategies and collusion in network games with splittable flow. In: Bampis, E., Skutella, M. (eds.) WAOA 2008. LNCS, vol. 5426, pp. 133–146. Springer, Heidelberg (2009)
8. Hayrapetyan, A., Tardos, E., Wexler, T.: The effect of collusion in congestion games. In: STOC, pp. 89–98 (2006)
9. Huang, C.-C.: The price of collusion in series-parallel networks, Technical Report 238, Humboldt-Universität zu Berlin, Institut für Informatik (2011)
10. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
11. Orda, A., Rom, R., Shimkin, N.: Competitive routing in multiuser communication networks. IEEE/ACM Transactions on Networking 1(5), 510–521 (1993)
12. Roughgarden, T.: Selfish routing with atomic players. In: SODA, pp. 1184–1185 (2005)
13. Roughgarden, T., Schoppmann, F.: Local smoothness and the price of anarchy in atomic splittable congestion game. In: SODA, pp. 255–267 (2011)
14. Roughgarden, T., Tardos, E.: How bad is selfish routing? Journal of the ACM 49(2), 236–259 (2002)
15. Wardrop, J.G.: Some theoretical aspects of road traffic research. In: Proc. Institute of Civil Engineers, Pt. II, vol. 1, pp. 325–378 (1952)

Privacy-Preserving Access of Outsourced Data via Oblivious RAM Simulation

Michael T. Goodrich[1] and Michael Mitzenmacher[2]

[1] University of California, Irvine
[2] Harvard University

Abstract. We describe schemes for the *oblivious RAM simulation* problem with a small logarithmic or polylogarithmic amortized increase in access times, with a very high probability of success, while keeping the external storage to be of size $O(n)$.

1 Introduction

Suppose Alice owns a large data set, which she outsources to an honest-but-curious server, Bob. For the sake of privacy, Alice can, of course, encrypt the cells of the data she stores with Bob. But encryption is not enough, as Alice can inadvertently reveal information about her data based on how she accesses it. Thus, we desire that Alice's access sequence (of memory reads and writes) is *data-oblivious*, that is, the probability distribution for Alice's access sequence should depend only on the size, n, of the data set and the number of memory accesses. Formally, we say a computation is data-oblivious if $\mathbf{Pr}(S \mid \mathcal{M})$, the probability that Bob sees an access sequence, S, conditioned on a specific configuration of his memory (Alice's outsourced memory), $\mathcal{M}$, satisfies $\mathbf{Pr}(S \mid \mathcal{M}) = \mathbf{Pr}(S \mid \mathcal{M}')$, for any memory configuration $\mathcal{M}' \neq \mathcal{M}$ such that $|\mathcal{M}'| = |\mathcal{M}|$. In particular, Alice's access sequence should not depend on the values of any set of memory cells in the outsourced memory that Bob maintains for Alice. To provide for full application generality, we assume outsourced data is indexed and we allow Alice to make arbitrary indexed accesses to this data for queries and updates. That is, let us assume this outsourced data model is as general as the random-access machine (RAM) model.

Most computations that Alice would be likely to perform on her outsourced data are not naturally data-oblivious. We are therefore interested in this paper in simulation schemes that would allow Alice to make her access sequence data-oblivious with low overhead. For this problem, which is known as *oblivious RAM simulation* [5], we are primarily interested in the case where Alice has a relatively small private memory, say, of constant size or size that is $O(n^{1/r})$, for a given constant $r > 1$.

Our Results. In this paper, we show how Alice can perform an oblivious RAM simulation, with very high probability[1], with an amortized time overhead of $O(\log n)$ and

[1] We show that our simulation fails to be oblivious with negligible probability; that is, the probability that the algorithm fails can be shown to be $O\left(\frac{1}{n^\alpha}\right)$ for any $\alpha > 0$. We say an event holds with *very high probability* if it fails with negligible probability.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 576–587, 2011.

with $O(n)$ storage overhead purchased from Bob, while using a private memory of size $O(n^{1/r})$, for a given constant $r > 1$. With a constant-sized memory, we show that she can do this simulation with overhead $O(\log^2 n)$, with a similarly high probability of success. At a high level, our result shows that Alice can leverage the privacy of her small memory to achieve privacy in her much larger outsourced data set of size n. Interestingly, our techniques involve the interplay of some seemingly unrelated new results, which may be of independent interest, including an efficient MapReduce parallel algorithm for cuckoo hashing and a novel deterministic data-oblivious external-memory sorting algorithm.

Previous Related Results. Goldreich and Ostrovsky [5] introduce the oblivious RAM simulation problem and show that it requires an overhead of $\Omega(\log n)$ under some reasonable assumptions about the nature of such simulations. For the case where Alice has only a constant-size private memory, they show how Alice can easily achieve an overhead of $O(n^{1/2} \log n)$, with $O(n)$ storage at Bob's server, and, with a more complicated scheme, how Alice can achieve an overhead of $O(\log^3 n)$ with $O(n \log n)$ storage at Bob's server.

Williams and Sion [15] study the oblivious RAM simulation problem for the case when the data owner, Alice, has a private memory of size $O(n^{1/2})$, achieving an expected amortized time overhead of $O(\log^2 n)$ using $O(n \log n)$ memory at the data provider. Incidentally, Williams *et al.* [16] claim an alternative method that uses an $O(n^{1/2})$-sized private memory and achieves $O(\log n \log \log n)$ amortized time overhead with a linear-sized outsourced storage, but some researchers (e.g., see [14]) have raised concerns with the assumptions and analysis of this result.

The results of this paper were posted by the authors in preliminary form as [7]. Independently, Pinkas and Reinman [14] published an oblivious RAM simulation result for the case where Alice maintains a constant-size private memory, claiming that Alice can achieve an expected amortized overhead of $O(\log^2 n)$ while using $O(n)$ storage space at the data outsourcer, Bob. Unfortunately, their construction contains a flaw that allows Bob to learn Alice's access sequence, with high probability, in some cases, which our construction avoids.

Ajtai [1] shows how oblivious RAM simulation can be done with a polylogarithmic factor overhead without cryptographic assumptions about the existence of random hash functions, as is done in previous papers [5,14,15,16], as well as any paper that derives its security or privacy from the random oracle model (including this paper). A similar result is also given by Damgård *et al.* [2]. Although these results address interesting theoretical limits of what is achievable without random hash functions, we feel that the assumption about the existence of random hash functions is actually not a major obstacle in practice, given the ubiquitous use of cryptographic hash functions.

2 Preliminaries

A Review of Cuckoo Hashing. Pagh and Rodler [13] introduce *cuckoo hashing*, which is a hashing scheme using two tables, each with m cells, and two hash functions, h_1 and h_2, one for each table, where we assume h_1 and h_2 can be modeled as random hash functions for the sake of analysis. The tables store $n = (1 - \epsilon)m$ keys, where one key

can be held in each cell, for a constant $\epsilon < 1$. Keys can be inserted or deleted over time; the requirement is that at most $n = (1-\epsilon)m$ distinct keys are stored at any time. A stored key x should be located at either $h_1(x)$ or $h_2(x)$, and, hence, lookups take constant time. On insertion of a new key x, cell $h_1(x)$ is examined. If this cell is empty, x is placed in this cell and the operation is complete. Otherwise, x is placed in this cell and the key y already in the cell is moved to $h_2(y)$. This may in turn cause another key to be moved, and so on. We say that a failure occurs if, for an appropriate constant c_0, after $c_0 \log n$ steps this process has not successfully terminated with all keys located in an appropriate cell. Suppose we insert an nth key into the system. Well-known attributes of cuckoo hashing include:

- The expected time to insert a new key is bounded above by a constant.
- The probability a new key causes a failure is $\Theta(1/n^2)$.

Kirsch, Mitzenmacher, and Wieder introduce the idea of utilizing a *stash* [10]. A stash can be thought of as an additional memory where keys that would otherwise cause a failure can be placed. In such a setting, a failure occurs only if the stash itself overflows. For n items inserted in a two-table cuckoo hash table, the total failure probability can be reduced to $O(1/n^{k+1})$ for any constant k using a stash that can hold k keys. For our results, we require a generalization of this result to stashes that are larger than constant sized.

3 MapReduce Cuckoo Hashing

The MapReduce Paradigm. In the MapReduce paradigm (e.g., see [4,9]), a parallel computation is defined on a set of values, $\{x_1, x_2, \ldots, x_n\}$, and consists of a series of *map*, *shuffle*, and *reduce* steps:

- A map step applies a *mapping function*, μ, to each value, x_i, to produce a key-value pair, (k_i, v_i). To allow for parallel execution, the function, $\mu(x_i) \rightarrow (k_i, v_i)$, must depend only on x_i.
- A shuffle step takes all the key-value pairs produced in the previous map step, and produces a set of lists, $L_k = (k; v_{i_1}, v_{i_2}, \ldots)$, where each such list consists of all the values, v_{i_j}, such that $k_{i_j} = k$ for a key k assigned in the map step.
- A reduce step applies a *reduction function*, ρ, to each list, $L_k = (k; v_{i_1}, v_{i_2}, \ldots)$, formed in the shuffle step, to produce a set of values, $w_1, w_2, \ldots$. The reduction function, ρ, is allowed to be defined sequentially on L_k, but should be independent of other lists $L_{k'}$ where $k' \neq k$.

Since we are using a MapReduce algorithm as a means to an end, rather than as an end in itself, we allow values produced at the end of a reduce step to be of two types: *final values*, which should be included in the output of such an algorithm when it completes and are not included in the set of values given as input to the next map step, and *non-final values*, which are to be used as the input to the next map step. Thus, for our purposes, a MapReduce computation continues performing map, shuffle, and reduce steps until the last reduce step is executed, at which point we output all the final values produced over the course of the algorithm.

In the MUD version of this model [4], which we call the *streaming-MapReduce* model, the computation of ρ is restricted to be a streaming algorithm that uses only $O(\log^c n)$ working storage, for a constant $c \geq 0$. Given our interest in applications to data-oblivious computations, we define a version that further restricts the computation of ρ to be a streaming algorithm that uses only $O(1)$ working storage. That is, we focus on a streaming-MapReduce model where $c = 0$, which we call the *sparse-streaming-MapReduce* model. In applying this paradigm to solve some problem, we assume we are initially given a set of n values as input, for which we then perform t steps of map-shuffle-reduce, as specified by a sparse-streaming-MapReduce algorithm, $\mathcal{A}$.

Let us define the *message complexity* of a MapReduce to be the total size of all the inputs and outputs to all the map, shuffle, and reduce steps in a MapReduce algorithm. That is, if we let n_i denote the total size of the input and output sets for the ith phase of map, shuffle, and reduce steps, then the message complexity of a MapReduce algorithm is $\sum_i n_i$.

Suppose we have a function, $f(i, n)$, such that $n_i \leq f(i, n)$, for each phase i, over all possible executions of a MapReduce algorithm, $\mathcal{A}$, that begins with an input of size n. In this case, let us say that f is a *ceiling function* for $\mathcal{A}$. Such a function is useful for bounding the worst-case performance overhead for a MapReduce computation.

A MapReduce Algorithm for Cuckoo Hashing. Let us now describe an efficient algorithm for setting up a cuckoo hashing scheme for a given set, $X = \{x_1, x_2, \ldots, x_n\}$, of items, which we assume come from a universe that can be linearly ordered in some arbitrary fashion. Let T_1 and T_2 be the two tables that we are to use for cuckoo hashing and let h_1 and h_2 be two candidate hash functions that we are intending to use as well.

For each x_i in X, recall that $h_1(x_i)$ and $h_2(x_i)$ are the two possible locations for x_i in T_1 and T_2. We can define a bipartite graph, G, commonly called the *cuckoo graph*, with vertex sets $U = \{h_1(x_i)\colon\ x_i \in X\}$ and $W = \{h_2(x_i)\colon\ x_i \in X\}$ and edge set $E = \{(h_1(x_i), h_2(x_i))\colon\ x_i \in X\}$. That is, for each edge (u, v) in E, there is an associated value x_i such that $(u, v) = (h_1(x_i), h_2(x_i))$, with parallel edges allowed. Imagine for a moment that an oracle identifies for us each connected component in G and labels each node v in G with the smallest item belonging to an edge of v's connected component. Then we could initiate a breadth-first search from the node u in U such that $h_1(x_i) = u$ and x_i is the smallest item in u's connected component, to define a BFS tree T rooted at u. For each non-root node v in T, we can store the item x_j at v, where x_j defines the edge from v to its parent in T.

If a connected component C in G is in fact a tree, then this breadth-first scheme will accommodate all the items associated with edges of C. Otherwise, if C contains some non-tree edges with respect to its BFS tree, then we pick one such edge, e. All other non-tree edges belong to items that are going to have to be stored in the stash. For the one chosen non-tree edge, e, we assign e's item to one of e's endvertices, w, and we perform a "cuckoo" action along the path, π, from w up to the root of its BFS tree, moving each item on π from its current node to the parent of this node on π. Therefore, we can completely accommodate all items associated with unicyclic subgraphs or tree subgraphs for their connected components. All other items are stored

in the stash. For cuckoo graphs corresponding to hash tables with load less than $1/2$, with high probability there are no components with two or more non-tree edges, and the stash further increases the probability that, when such edges exist, they can be handled.

Unfortunately, we don't have an oracle to initiate the above algorithm. Instead, we essentially perform the above algorithm in parallel, starting from all nodes in U, assuming they are the root of a BFS tree. Whenever we discover a node should belong to a different BFS tree, we simply ignore all the work we did previously for this node and continue the computation for the "winning" BFS tree (based on the smallest item in that connected component). Consider an efficient MapReduce algorithm for performing n simultaneous breadth-first searches such that, any time two searches "collide," the search that started from a lower-numbered vertex is the one that succeeds. We can easily convert this into an algorithm for cuckoo hashing by adding steps that process non-tree edges in a BFS search. For the first such edge we encounter, we initiate a reverse cuckoo operation, to allocate items in the induced cycle. For all other non-tree edges, we allocate their associated items to the stash.

Intuitively, the BFS initiated from the minimum-numbered vertex, v, in a connected component propagates out in a wave, bounces at the leaves of this BFS tree, returns back to v to confirm it as the root, and then propagates back down the BFS tree to finalize all the members of this BFS tree. Thus, in time proportional to the depth of this BFS tree (which, in turn, is at most the size of this BFS tree), we will finalize all the members of this BFS tree. And once these vertices are finalized, we no longer need to process them any more. Moreover, this same argument applies to the modified BFS that performs the cuckoo actions. Therefore, we process each connected component in the cuckoo graph in a number of iterations that is, in the worst-case, equal to three times the size of each such component (since the waves of the BFS move down-up-down, passing over each vertex three times).

To bound both the time for the parallel BFS algorithm to run and to bound its total work, we require bounds on the component sizes that arise in the cuckoo graph. Such bounds naturally appear in previous analyses of cuckoo hashing. In particular, the following result is proven in [10][Lemma 2.4].

Lemma 1. *Let v be any fixed vertex in the cuckoo graph and let C_v be its component. Then there exists a constant $\beta \in (0,1)$ such that for any integer $k \geq 0$,*

$$\mathbf{Pr}(|C_v| \geq k) \leq \beta^k.$$

More detailed results concerning the asymptotics of the distribution of component sizes for cuckoo hash tables can be found in, for example [3], although the above result is sufficient to prove linear message-complexity overhead.

Lemma 1 immediately implies that the MapReduce BFS algorithm (and the extension to cuckoo hashing) takes $O(\log n)$ time to complete with high probability.

Lemma 2. *The message complexity of the MapReduce BFS algorithm is $O(n)$ with very high probability.*

Proof. The message complexity is bounded by a constant times $\sum_v |C_v|$, which in expectation is

$$\mathbf{E}\left[\sum_v |C_v|\right] = \sum_v \mathbf{E}[|C_v|] \leq 2m \sum_{k\geq 0} \mathbf{Pr}(C_v \geq k) \leq 2m \sum_{k\geq 0} \beta^k = O(m).$$

To prove a very high probability bound, we use a variant of Azuma's inequality specific to this situation. If all component sizes were bounded by say $O(\log^2 n)$, then a change in any single edge in the cuckoo graph could affect $\sum_v |C_v|$ by only $O(\log^4 n)$, and we could directly apply Azuma's inequality to the Doob martingale obtained by exposing the edges of the cuckoo graph one at a time. Unfortunately, all component sizes are $O(\log^2 n)$ only with very high probability. However, standard results yield that one can simply add in the probability of a "bad event" to a suitable tail bound, in this case the bad event being that some component size is larger than $c_1 \log^2 n$ for some suitable constant c_1. Specifically, we directly utilize Theorem 3.7 from [12], which allows us to conclude that if the probability of a bad event is a superpolynomially small δ, then

$$\mathbf{Pr}\left(\sum_v |C_v| \geq \sum_v \mathbf{E}[|C_v|] + \lambda\right) \leq e^{-(2\lambda^2)/(nc_2 \log^4 n)} + \delta,$$

where c_2 is again a suitably chosen constant. Now choosing $\lambda = n^{2/3}$, for example, suffices. □

4 Simulating a MapReduce Algorithm Obliviously

Our simulation is based on a reduction to oblivious sorting.

Theorem 1. *Suppose $\mathcal{A}$ is a sparse-streaming-MapReduce algorithm that runs in at most t map-shuffle-reduce steps, and suppose further that we have a ceiling function, f, for $\mathcal{A}$. Then we can simulate $\mathcal{A}$ in a data-oblivious fashion in the RAM model in time $O(\sum_{i=1}^{t}$ o-sort$(f(i,n)))$, where o-sort(n) is the time needed to sort n items in a data-oblivious fashion.*

Proof. Let us consider how we can simulate the map, shuffle, and reduce steps in phase i of algorithm $\mathcal{A}$ in a data-oblivious way. We assume inductively that we store the input values for phase i in an array, X_i. Let us also assume inductively that X_i can store values that were created as final values the step $i-1$. A single scan through the first $f(i,n)$ values of X_i, applying the map function, μ, as we go, produces all the key-value pairs for the map step in phase i (where we output a dummy value for each input value that is final or is itself a dummy value). We can store each computed value, one by one, in an oblivious fashion using an output array Y. We then obliviously sort Y by keys to bring together all key-value pairs with the same key as consecutive cells in Y (with dummy values taken to be larger than all real keys). This takes time $O($o-sort$(f(i,n))$. Let us then do a scan of the first $f(i,n)$ cells in Y to simulate the reduce step. As we consider each item z in Y, we can keep a constant number of state variables as registers in our

RAM, which collectively maintain the key value, k, we are considering, the internal state of registers needed to compute ρ on z, and the output values produced by ρ on z. This size bound is due to the fact that $\mathcal{A}$ is a sparse-streaming-MapReduce algorithm. Since the total size of this state is constant, the total number of output values that could possibly be produced by ρ on an input z can be determined a priori and bounded by a constant, d. So, for each value z in Y, we write d values to an output array Z, according to the function ρ, padding with dummy values if needed. The total size of Z is therefore $O(d\, f(i,n))$, which is $O(f(i,n))$. Still, we cannot simply make Z the input for the next map-shuffle-reduce step at this point, since we need the input array to have at most $f(i,n)$ values. Otherwise, we would have an input array that is a factor of d too large for the next phase of the algorithm $\mathcal{A}$. So we perform a data-oblivious sorting of Z, with dummy values taken to be larger than all real values, and then we copy the first $f(i,n)$ values of Z to X_{i+1} to serve as the input array for the next step to continue the inductive argument. The total time needed to perform step i is $O(\text{o-sort}(f(i,n))$. When we have completed processing of step t, we concatenate all the X_i's together, flagging all the final values as being the output values for the algorithm $\mathcal{A}$, which can be done in a single data-oblivious scan. Therefore, we can simulate each step of $\mathcal{A}$ in a data-oblivious fashion and produce the output from $\mathcal{A}$, as well, at the end. Since we do two sorts on arrays of size $O(f(i,n))$ in each map-shuffle-reduce step, i, of $\mathcal{A}$, this simulation takes time $O(\sum_{i=1}^{t} \text{o-sort}(f(i,n)))$. □

We can show that by combining this result with Lemma 2 we get the following:

Theorem 2. *Given a set of n distinct items and corresponding hash values, there is a data-oblivious algorithm for constructing a two-table cuckoo hashing scheme of size $O(n)$ with a stash of size s, whenever this stash size is sufficient, in $O(\text{o-sort}(n+s))$ time.*

External-Memory Data-Oblivious Sorting. In this section, we give our efficient external-memory oblivious sorting algorithm. Recall that in this model memory is divided between an internal memory of size M and an external memory (like a disk), which initially stores an input of size N, and that the external memory is divided into blocks of size B, for which we can read or write any block in an atomic action called an I/O. In this context, we say that an external-memory sorting algorithm is *data-oblivious* if the sequence of I/Os that it performs is independent of the values of the data it is processing. So suppose we are given an unsorted array A of N comparable items stored in external memory. If $N \leq M$, then we copy A into our internal memory, sort it, and copy it back to disk. Otherwise, we divide A into $k = \lceil (M/B)^{1/3} \rceil$ subarrays of size N/k and recursively sort each subarray. Thus, the remaining task is to merge these subarrays into a single sorted array.

Let us therefore focus on the task of merging k sorted arrays of size $n = N/k$ each. If $nk \leq M$, then we copy all the lists into internal memory, merge them, and copy them back to disk. Otherwise, let $A[i,j]$ denote the jth element in the ith array. We form a set of m new subproblems, where the pth subproblem involves merging the k sorted subarrays defined by $A[i,j]$ elements such that $j \bmod m = p$, for $m = \lceil (M/B)^{1/3} \rceil$. We form these subproblems by processing each input subarray and filling in the portions of the output subarrays from the input, sending full blocks to disk when they fill up

(which will happen at deterministic moments independent of data values), all of which uses $O(N/B)$ I/Os. Then we recursively solve all the subproblems. Let $D[i, j]$ denote the jth element in the output of the ith subproblem. That is, we can view D as a two-dimensional array, with each row corresponding to the solution to a recursive merge.

Lemma 3. *Each row and column of D is in sorted order and all the elements in column j are less than or equal to every element in column $j + k$.*

Proof. The lemma follows from Theorem 1 of Lee and Batcher [11]. □

To complete the k-way merge, then, we imagine that we slide an $m \times k$ rectangle across D, from left to right. We begin by reading into internal memory the first $2k$ columns of D. Next, we sort this set of elements in internal memory and we output the ordered list of the km smallest elements (holding back a small buffer of elements if we don't fill up the last block). Then we read in the next k columns of D (possibly reading in k additional blocks for columns beyond this, depending on the block boundaries), and repeat the sorting of the items in internal memory and outputting the smallest km elements in order. At any point in this algorithm, we may need to have up to $2km + (m+2)B$ elements in internal memory, which, under a reasonable tall cache assumption (say $M > 3B^4$), will indeed fit in internal memory. We continue in this way until we process all the elements in D. Note that, since we process the items in D from left to right in a block fashion, for all possible data values, the algorithm is data-oblivious with respect to I/Os.

Consider the correctness of this method. Let $D_1, D_2, \ldots, D_l$ denote the subarrays of D of size $m \times k$ used in our algorithm. By a slight abuse of notation, we have that $D_1 \leq D_3$, by Lemma 3. Thus, the smallest mk items in $D_1 \cup D_2$ are less than or equal to the items in D_3. Likewise, these mk items are obviously less than the largest mk items in $D_1 \cup D_2$. Therefore, the first mk items output by our algorithm are the smallest mk items in D. Inductively, then, we can now ignore these smallest mk items and repeat this argument with the remaining items in D. Thus, we have the following.

Theorem 3. *Given an array A of size N comparable items, we can sort A with a data-oblivious external-memory algorithm that uses $O((N/B)\log^2_{M/B}(N/B))$ I/Os, under a tall-cache assumption ($M > 3B^4$).*

Theorem 4. *Given a set of n distinct items and corresponding hash values, there is a data-oblivious algorithm for constructing a two-table cuckoo hashing scheme of size $O(n)$ with a stash of size $s = O(\log n)$ whenever this stash size is sufficient, using a private memory of size $O(n^{1/r})$, for a given fixed constant $r > 1$, in $O(n + s)$ time.*

Proof. Combine Theorems 2 and 3, with $N = n + s$, $B = 1$, and $M \in O(n^{1/r})$. □

5 Oblivious RAM Simulations

Our data-oblivious simulation of a non-oblivious algorithm on a RAM follows the general approach of Goldreich and Ostrovsky [5], but differs from it in some important

ways, most particularly in our use of cuckoo hashing. We assume throughout that Alice encrypts the data she outsources to Bob using a probabilistic encryption scheme, so that multiple encryptions of the same value are extremely likely to be different. Thus, each time she stores an encrypted value, there is no way for Bob to correlate this value to other values or previous values. So the only remaining information that needs to be protected is the sequence of accesses that Alice makes to her data.

Our description simultaneously covers two separate cases: the constant-sized private memory case with very high probability amortized time bounds, and the case of private memory of size $O(n^{1/r})$ for some constant $r > 1$. The essential description is the same for these settings, with slight differences in how the hash tables are structured as described below.

We store the n data items in a hierarchy of hash tables, H_k, H_{k+1}, ..., H_L, where k is an initial starting point for our hierarchy and $L = \log n$. Each table, H_i, has capacity for 2^i items, which are distributed between "real" items, which correspond to memory cells of the RAM, plus "dummy" items, which are added for the sake of obliviousness to make the number of items stored in H_i be the appropriate value. The starting table, H_k, is simply an array that we access exhaustively with each RAM memory read or write. The lower-level tables, H_{k+1} to H_l, for l determined in the analysis, are standard hash tables with H_i having 2^{i+1} buckets of size $O(\log n)$, whereas higher-level tables, H_{l+1} to H_L, are cuckoo hash tables, with H_i having $(1+\epsilon)2^{i+2}$ cells and a stash of size s, where s is determined in the analysis and $\epsilon > 0$ is a constant. The sizes of the hash tables in this hierarchy increase geometrically; hence the total size of all the hash tables is proportional to the size of H_L, which is $O(n)$. Our two settings will differ in the starting points for the various types of hash tables in the hierarchy as well as the size of the stash associated with the hash tables.

For each H_i with $i < L$ we keep a count, d_i, of the number of times that H_i has been accessed since first being constructed as an "empty" hash table containing 2^i dummy values, numbered consecutively from -1 to -2^i. For convenience, in what follows, let us think of each hash table H_i with $i > l$ as being a standard cuckoo hash table, with a stash of size $s = \Theta(\log n)$ chosen for the sake of a desired superpolynomially-small error probability. Initially, every H_i is an empty cuckoo hash table, except for H_L, which contains all $n = 2^L$ initial values plus 2^L dummy values.

We note that there is, unfortunately, some subtlety in the geometric construction of hash tables in the setting of constant-sized private memory with very high probability bounds. A problem arises in that for small hash tables, of size say polylogarithmic in n, it is not clear that appropriate very high probability bounds, which required failures to occur with probability inverse superpolynomial in n, hold with logarithmic sized stashes. Such results do not follow from previous work [10], which focused on constant-sized stashes. However, to keep the simulation time small, we cannot have larger than a logarithmic-sized stash (if we are searching it exhaustively), and we require small initial levels to keep the simulation time small.

We can show that we can cope with the problem by extending results from [10] that logarithmic sized stashes are sufficient to obtain the necessary probability bounds for hash tables of size that are polylogarithmic in n. In order to start our hierarchy with

small, logarithmic-sized hash tables, we simply use standard hash tables as described above for levels $k+1$ to $l = O(\log \log n)$ and use cuckoo hash tables, each with a stash of size $s = O(\log n)$, for levels $l+1$ to L.

Access Phase. When we wish to make an access for a data cell at index x, for a read or write, we first look in H_k exhaustively to see if it contains an item with key x. Then, we initialize a flag, "found," to **false** iff we have not found x yet. We continue by performing an access[2] to each hash table H_{k+1} to H_L, which is either to x or to a dummy value (if we have already found x).

Our privacy guarantee depends on us never repeating a search. That is, we never perform a lookup for the same index, x or d, in the same table, that we have in the past. Thus, after we have performed the above lookup for x, we add x to the table H_k, possibly with a new data value if the access action we wanted to perform was a write.

Rebuild Phase. With every table H_i, we associate a *potential*, p_i. Initially, every table has zero potential. When we add an element to H_k, we increment its potential. When a table H_i has its potential, p_i, reach 2^i, we reset $p_i = 0$ and empty the unused values in H_i into H_{i+1} and add 2^i to p_{i+1}. There are at most 2^i such unused values, so we pad this set with dummy values to make it be of size exactly 2^i; these dummy values are included for the sake of obliviousness and can be ignored after they are added to H_{i+1}. Of course, this could cause a cascade of emptyings in some cases, which is fine.

Once we have performed all necessary emptyings, then for any $j < L$, $\sum_{i=1}^{j} p_i$ is equal to the number of accesses made to H_j since it was last emptied. Thus, we rehash each H_i after it has been accessed 2^i times. Moreover, we don't need to explicitly store p_i with its associated hash table, H_i, as d_i can be used to infer the value of p_i.

The first time we empty H_i into an empty H_{i+1}, there must have been exactly 2^i accesses made to H_{i+1} since it was created. Moreover, the first emptying of H_i into H_{i+1} involves the addition of 2^i values to H_{i+1}, some of which may be dummy values.

When we empty H_i into H_{i+1} for the second time it will actually be time to empty H_{i+1} into H_{i+2}, as H_{i+1} would have been accessed 2^{i+1} times by this point—so we can simply union the current (possibly padded) contents of H_i and H_{i+1} together to empty both of them into H_{i+2} (possibly with further cascaded emptyings). Since the sizes of hash tables in our hierarchy increase geometrically, the size of our final rehashing problem will always be proportional to the size of the final hash table that we want to construct. Every $n = 2^L$ accesses we reconstruct the entire hierarchy, placing all the current values into H_L. Thus, the schedule of table emptyings follows a data-oblivious pattern depending only on n.

Correctness and Analysis. To the adversary, Bob, each lookup in a hash table, H_i, is to one random location (if the table is a standard hash table) or to two random locations and the s elements in the stash (if the table is a cuckoo hash table), which can be

1. a search for a real item, x, that is not in H_i,
2. a search for a real item, x, that is in H_i,
3. a search for a dummy item, d_i.

[2] This access is actually two accesses if the table is a cuckoo hash table.

Moreover, as we search through the levels from k to L we go through these cases in this order (although for any access, we might not ever enter case 1 or case 3, depending on when we find x). In addition, if we search for x and don't find it in H_i, we will eventually find x in some H_j for $j > i$ and then insert x in H_k; hence, if ever after this point in time we perform a future search for x, it will be found prior to H_i. In other words, we will never repeat a search for x in a table H_i. Moreover, we continue performing dummy lookups in tables H_j, for $j > i$, even after we have found the item for cell x in H_i, which are to random locations based on a value of d_i that is also not repeated. Thus, the accesses we make to any table H_i are to locations that are chosen independently at random. In addition, so long as our accesses don't violate the possibility of our tables being used in a valid cuckoo hashing scheme (which our scheme guarantees with very high probability) then all accesses are to independent random locations that also happen to correspond to locations that are consistent with a valid cuckoo scheme. Finally, note that we rebuild each hash table H_i after we have made 2^i accesses to it. Of course, some of these 2^i accesses may have successfully found their search key, while others could have been for dummy values or for unsuccessful searches. Nevertheless, the collection of 2^i distinct keys used to perform accesses to H_i will either form a standard hash table or a cuckoo graph that supports a cuckoo hash table, with a stash of size s, w.v.h.p. Therefore, with very high probability, the adversary will not be able to determine which among the search keys resulted in values that were found in H_i, which were to keys not found in H_i, and which were to dummy values.

Each memory access involves at most $O(s \log n)$ reads and writes, to the tables H_k to H_L. In addition, note that each time an item is moved into H_k, either it or a surrogate dummy value may eventually be moved from H_k all the way to H_L, participating in $O(\log n)$ rehashings, with very high probability. In the constant-memory case, by Theorem 2, each data-oblivious rehashing of n_i items takes $O((n_i+s)\log(n_i+s))$ time. In addition, in this case, we use a stash of size $s \in O(\log n)$ and set $l = k + O(\log \log n)$. In the case of a private memory of size $O(n^{1/r})$, each data-oblivious rehashing of n_i items takes $O(n_i)$ time, by Theorem 4. In addition, in this case, we can use a constant-size stash (i.e., $s = O(1)$), but start with $k = (1/r)\log n$, so that H_k fits in private memory (with all the other H_i's being in the outsourced memory). The use of a constant-sized stash, however, limits us to a result that holds with high probability, instead of with very high probability.

To achieve very high probability in this latter case, we utilize a technique suggested in [6]. Instead of using a constant-sized stash in each level, we combine them into a single logarithmic-sized stash, used for all levels. This allows the stash at any single level to possibly be larger than any fixed constant, giving bounds that hold with very high probability. Instead of searching the stash at each level, Alice must load the entire stash into private memory and rewrite it on each memory access. Therefore, we have the following.

Theorem 5. *Data-oblivious RAM simulation of a memory of size n can be done in the constant-size private-memory case with an amortized time overhead of $O(\log^2 n)$, with very high probability. Such a simulation can be done in the case of a private memory of size $O(n^{1/r})$ with an amortized time overhead of $O(\log n)$, with very high probability, for a constant $r > 1$. The space needed at the server in all cases is $O(n)$.*

Acknowledgments. We thank Radu Sion and Peter Williams for helpful discussions about oblivious RAM simulation. This research was supported in part by the National Science Foundation under grants 0724806, 0713046, 0847968, 0915922, 0953071, and 0964473. A full version of this paper is available as [8].

References

1. Ajtai, M.: Oblivious RAMs without cryptographic assumptions. In: Proc. of the 42nd ACM Symp. on Theory of Computing (STOC), pp. 181–190. ACM, New York (2010)
2. Damgård, I., Meldgaard, S., Nielsen, J.B.: Perfectly secure oblivious RAM without random oracles. Cryptology ePrint Archive, Report 2010/108 (2010), http://eprint.iacr.org/
3. Drmota, M., Kutzelnigg, R.: A precise analysis of cuckoo hashing (2008) (preprint), http://www.dmg.tuwien.ac.at/drmota/cuckoohash.pdf
4. Feldman, J., Muthukrishnan, S., Sidiropoulos, A., Stein, C., Svitkina, Z.: On distributing symmetric streaming computations. In: ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 710–719 (2008)
5. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM 43(3), 431–473 (1996)
6. Goodrich, M., Mitzenmacher, M., Ohrimenko, O., Tamassia, R.: Privacy-preserving group data access via stateless oblivious RAM simulation (unpublished manuscript)
7. Goodrich, M.T., Mitzenmacher, M.: MapReduce Parallel Cuckoo Hashing and Oblivious RAM Simulations. ArXiv e-prints, Eprint 1007.1259v1 (July 2010)
8. Goodrich, M.T., Mitzenmacher, M.: Privacy-Preserving Access of Outsourced Data via Oblivious RAM Simulation. ArXiv e-prints, Eprint 1007.1259v2 (April 2011)
9. Karloff, H., Suri, S., Vassilvitskii, S.: A model of computation for MapReduce. In: Proc. ACM-SIAM Sympos. Discrete Algorithms (SODA), pp. 938–948 (2010)
10. Kirsch, A., Mitzenmacher, M., Wieder, U.: More robust hashing: cuckoo hashing with a stash. SIAM J. Comput. 39, 1543–1561 (2009)
11. Lee, D.-L., Batcher, K.E.: A multiway merge sorting network. IEEE Trans. on Parallel and Distributed Systems 6(2), 211–215 (1995)
12. McDiarmid, C.: Concentration. In: Habib, M., McDiarmid, C., Ramirez-Alfonsin, J., Reed, B. (eds.) Probabilistic Methods for Algorithmic Discrete Mathematics, pp. 195–248. Springer, Berlin (1998)
13. Pagh, R., Rodler, F.: Cuckoo hashing. Journal of Algorithms 52, 122–144 (2004)
14. Pinkas, B., Reinman, T.: Oblivious RAM revisited. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 502–519. Springer, Heidelberg (2010)
15. Williams, P., Sion, R.: Usable pir. In: NDSS. The Internet Society, San Diego (2008)
16. Williams, P., Sion, R., Carbunar, B.: Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In: 15th ACM Conf. on Computer and Communications Security (CCS), pp. 139–148 (2008)

Adaptively Secure Non-interactive Threshold Cryptosystems

Benoît Libert[1,⋆] and Moti Yung[2]

[1] Université catholique de Louvain, ICTEAM Institute, Belgium
[2] Google Inc. and Columbia University, USA

Abstract. Threshold cryptography aims at enhancing the availability and security of decryption and signature schemes by splitting private keys into several (say n) shares (typically, each of size comparable to the original secret key). In these schemes, a quorum of at least ($t \leq n$) servers needs to act upon a message to produce the result (decrypted value or signature), while corrupting less than t servers maintains the scheme's security. For about two decades, extensive study was dedicated to this subject, which created a number of notable results. So far, most practical threshold signatures, where servers act non-interactively, were analyzed in the limited static corruption model (where the adversary chooses which servers will be corrupted at the system's initialization stage). Existing threshold encryption schemes that withstand the strongest combination of adaptive malicious corruptions (allowing the adversary to corrupt servers at any time based on its complete view), and chosen-ciphertext attacks (CCA) all require interaction (in the non-idealized model) and attempts to remedy this problem resulted only in relaxed schemes. The same is true for threshold signatures secure under chosen-message attacks (CMA).

To date (for about 10 years), it has been open whether there are non-interactive threshold schemes providing the highest security (namely, CCA-secure encryption and CMA-secure signature) with scalable shares (*i.e.*, as short as the original key) and adaptive security. This paper answers this question affirmatively by presenting such efficient decryption and signature schemes within a unified algebraic framework.

Keywords: Threshold cryptography, encryption schemes, digital signatures, adaptive corruptions, non-interactivity.

1 Introduction

Threshold cryptography [17,18,7] avoids single points of failure by splitting cryptographic keys into $n > 1$ shares which are stored by servers in distinct locations. Cryptographic schemes are then designed in such a way that at least t out of n servers should contribute to private key operations in order for these to succeed. In (t, n)-threshold cryptosystems (resp. signature schemes), an adversary

⋆ This author acknowledges the Belgian National Fund For Scientific Research (F.R.S.-F.N.R.S.) for his "Chargé de Recherches" fellowship and the BCRYPT Interuniversity Attraction Pole.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 588–600, 2011.

breaking into up to $t-1$ servers should be unable to decrypt ciphertexts (resp. generate signatures) on her own.

Designing secure threshold public key schemes has proved to be a highly non-trivial task. For example, the random oracle model [2] was needed to analyze the first chosen-ciphertext secure (or CCA-secure for short) threshold encryption systems put forth by Shoup and Gennaro [35]. Canetti and Goldwasser [11] gave a standard model implementation (based on the Cramer-Shoup encryption scheme [12]) at the expense of using interaction between decryption servers to obtain robustness (*i.e.*, ensure that no dishonest minority deviating from the protocol can prevent uncorrupted servers from successfully decrypting). Other CCA-secure threshold cryptosystems were suggested in [31,19,4].

Non-Interactive Schemes. Using the innovative Canetti-Halevi-Katz (CHK) methodology [13], Boneh, Boyen and Halevi [4] showed the first *non-interactive* robust CCA-secure threshold cryptosystem with a security proof in the standard model (*i.e.*, without the random oracle idealization): in their scheme, decryption servers can compute their partial decryption result (termed "decryption share") *without* having to talk to each other and decryption shares contain built-in proofs of their validity, which guarantees robustness. Similar applications of the CHK methodology were studied in [8,27].

In the context of digital signatures, Shoup [36] described non-interactive threshold signatures based on RSA and providing robustness.

Adaptive Corruptions. Historically, threshold primitives (including [35,11, 19,23,4]) have been mostly studied in a static corruption model, where the adversary chooses which servers she wants to corrupt *before* the scheme is set up. Unfortunately, adaptive adversaries – who can choose whom to corrupt at any time and depending on the previously collected information – are known (see, e.g., [14]) to be strictly stronger and substantially harder to deal with.

To address the above concerns, Canetti *et al.* [10] proposed a method to cope with adaptive corruptions assuming reliable erasures (*i.e.*, players must be able to safely erase their local data when they no longer need them). Their techniques were used in [1] to build adaptively secure proactive RSA signatures. As a disadvantage, this approach requires all servers to refresh their shares in a proactive manner [32] (by jointly computing a sharing of zero) after each distributed private key operation (effectively making schemes n-out-of-n rather than t-out-of-n for any $t \leq n$). At the same time, Frankel, MacKenzie and Yung [24,25] showed how to avoid the latter limitation while still using erasures.

Later on, Jarecki and Lysyanskaya [26] eliminated the need for erasures and gave an adaptively secure variant of the Canetti-Goldwasser CCA-secure threshold cryptosystem [11]. Unfortunately, their scheme – which is also designed to remain secure in concurrent environments – requires a lot of interaction between decryption servers (at least in the standard model[1]).

[1] In the random oracle model, decryption servers can non-interactively prove the validity of their decryption shares using the Fiat-Shamir heuristic but only heuristic arguments are then possible in terms of security (see [9], for instance).

Recently, Qin *et al.* [33] suggested a non-interactive threshold cryptosystem (more precisely, a threshold broadcast encryption scheme whose syntax is similar to [15,16]) with adaptive security. Its downside is its lack of scalability since private key shares consist of $O(n)$ elements, where n is the number of servers (while, in prior schemes, the share size only depends on the security parameter).

Our contribution. We give the first robust threshold cryptosystem which is simultaneously chosen-ciphertext secure under adaptive corruptions and non-interactive while being scalable (*i.e.*, providing short private keys). Unlike [33], our scheme features constant-size private key shares (where "constant" means independent of t and n) for public keys of comparable size. In addition, it is conceptually simple and relies on assumptions of constant-size whereas [33] relies on a "q-type" assumption where the input is a sequence of the form $(g, g^{\alpha}, \ldots, g^{(\alpha^q)})$, for some secret $\alpha \in \mathbb{Z}_p$. Unlike [10], we do not have to perform proactive refreshes of private key shares after each decryption operation.

Our starting point is the identity-based encryption (IBE) system [5,34] proposed by Lewko and Waters [29] and the elegant dual system approach introduced by Waters [37]. The latter has proved useful to demonstrate full security in identity and attribute-based encryption [37,28,29] but, to the best of our knowledge, it has not been applied to threshold cryptosystems so far. It is worth noting that the security proof of our scheme is not simply a direct consequence of applying the CHK paradigm to the Lewko-Waters results [29] as the treatment of adaptive corruptions does not follow from [13,29]. Like [29], our proof uses a sequence of games. While we also use so-called semi-functional decryption shares and ciphertexts as in the IBE setting [29], we have to consider two distinct kinds of semi-functional ciphertexts and an additional step (which aims at making all private key shares semi-functional) is needed in the proof to end up in a game where proving the security is made simple.

We also describe a non-interactive threshold signature that follows the same line of development and which can be proven secure in the standard model under adaptive corruptions. This appears to be the first security result under adaptive corruptions for non-interactive threshold signatures in the standard model.

Technically speaking, the encryption scheme can be visualized as a variant of the Boneh-Boyen-Halevi threshold system [4] in groups whose order is a product $N = p_1p_2p_3$ of three primes, which are chosen at key generation. Interestingly, if the factorization of N is somehow leaked, the proof of security under static corruptions implied by [4] still applies and only the proof of adaptive security ceases to go through. We also believe the semantically-secure variant of our scheme (which is obtained by removing the appropriate "checksum values" allowing to hedge against chosen-ciphertext attacks) to be of interest in its own right since it is multiplicatively homomorphic (like the ElGamal encryption scheme [20]) and retains security under adaptive corruptions in the threshold setting. It can thus find applications in important protocols such as e-voting for example.

Organization. Section 2 recalls the definitions of threshold cryptosystems. The scheme and its CCA-security are analyzed in sections 3.1 and 3.2, respectively. Our threshold signature is presented in the full version of the paper.

2 Background and Definitions

2.1 Definitions for Threshold Public Key Encryption

Definition 1. *A non-interactive (t, n)-threshold encryption scheme is a set of algorithms with the following specifications.*

Setup(λ, t, n)**:** takes as input a security parameter λ and integers $t, n \in \mathsf{poly}(\lambda)$ (with $1 \leq t \leq n$) denoting the number of decryption servers n and the decryption threshold t. It outputs a triple $(PK, \mathbf{VK}, \mathbf{SK})$, where PK is the public key, $\mathbf{SK} = (SK_1, \ldots, SK_n)$ is a vector of n private-key shares and $\mathbf{VK} = (VK_1, \ldots, VK_n)$ is the corresponding vector of verification keys. Decryption server i is given the share (i, SK_i) that allows deriving decryption shares for any ciphertext. For each $i \in \{1, \ldots, n\}$, the verification key VK_i will be used to check the validity of decryption shares generated using SK_i.

Encrypt(PK, M)**:** is a randomized algorithm that, given a public key PK and a plaintext M, outputs a ciphertext C.

Ciphertext-Verify(PK, C)**:** takes as input a public key PK and a ciphertext C. It outputs 1 if C is deemed valid w.r.t. PK and 0 otherwise.

Share-Decrypt(PK, i, SK_i, C)**:** on input of a public key PK, a ciphertext C and a private-key share (i, SK_i), this (possibly randomized) algorithm outputs a special symbol $(i, \perp)$ if **Ciphertext-Verify**$(PK, C) = 0$. Otherwise, it outputs a decryption share $\mu_i = (i, \hat{\mu}_i)$.

Share-Verify(PK, VK_i, C, μ_i)**:** takes as input PK, the verification key VK_i, a ciphertext C and a purported decryption share $\mu_i = (i, \hat{\mu}_i)$. It outputs either 1 or 0. In the former case, μ_i is said to be a *valid* decryption share. We adopt the convention that $(i, \perp)$ is an invalid decryption share.

Combine$(PK, \mathbf{VK}, C, \{\mu_i\}_{i \in S})$**:** given PK, $\mathbf{VK}$, C and a subset $S \subset \{1, \ldots, n\}$ of size $t = |S|$ with decryption shares $\{\mu_i\}_{i \in S}$, this algorithm outputs either a plaintext M or $\perp$ if the set contains invalid decryption shares.

Chosen-ciphertext security. We use a definition of chosen-ciphertext security which is identical to the one of [35,4] with the difference that the adversary can adaptively choose which parties she wants to corrupt.

Definition 2. *A non-interactive (t, n)-Threshold Public Key Encryption scheme is secure against chosen-ciphertext attacks (or IND-CCA2 secure) and adaptive corruptions if no PPT adversary has non-negligible advantage in this game:*

1. *The challenger runs* **Setup**(λ, t, n) *to obtain PK,* $\mathbf{SK} = (SK_1, \ldots, SK_n)$ *and* $\mathbf{VK} = (VK_1, \ldots, VK_n)$*. It gives PK and* $\mathbf{VK}$ *to the adversary $\mathcal{A}$ and keeps* $\mathbf{SK}$ *to itself.*
2. *The adversary $\mathcal{A}$ adaptively makes the following kinds of queries:*
   - *Corruption query: $\mathcal{A}$ chooses $i \in \{1, \ldots, n\}$ and obtains SK_i.*
   - *Decryption query: $\mathcal{A}$ chooses an index $i \in \{1, \ldots, n\}$ and a ciphertext C. The challenger replies with $\mu_i =$* **Share-Decrypt**(PK, i, SK_i, C)*.*

3. *$\mathcal{A}$ chooses two equal-length messages M_0, M_1. The challenger flips a fair coin $\beta \stackrel{R}{\leftarrow} \{0,1\}$ and computes $C^\star = \mathbf{Encrypt}(PK, M_\beta)$.*
4. *$\mathcal{A}$ makes further queries as in step 2 but she is not allowed to make decryption queries on the challenge ciphertext $C^\star$.*
5. *$\mathcal{A}$ outputs a bit β' and is deemed successful if (i) $\beta' = \beta$; (ii) no more than $t-1$ private key shares were obtained by $\mathcal{A}$ (via corruption queries) in the whole game. As usual, $\mathcal{A}$'s advantage is $Adv(\mathcal{A}) = |\Pr[\mathcal{A}\ wins\ ] - \frac{1}{2}|$.*

Consistency. A (t,n)-Threshold Encryption scheme provides decryption consistency if no PPT adversary has non-negligible advantage in a three-stage game where stages 1 and 2 are identical to those of definition 2. In stage 3, the adversary outputs a ciphertext C and two t-sets of decryption shares $\Phi = \{\mu_1, \ldots, \mu_t\}$ and $\Phi' = \{\mu'_1, \ldots, \mu'_t\}$. The adversary $\mathcal{A}$ is declared successful if

1. $\mathbf{Ciphertext\text{-}Verify}(PK, C) = 1$.
2. Φ and Φ' only consist of valid decryption shares.
3. $\mathbf{Combine}(PK, \mathbf{VK}, C, \Phi) \neq \mathbf{Combine}(PK, \mathbf{VK}, C, \Phi')$.

We note that condition 1 aims at preventing an adversary from trivially winning by outputting an invalid ciphertext, for which distinct sets of key shares may give different results. This definition of consistency is identical to the one of [35,4] with the difference that $\mathcal{A}$ can adaptively corrupt decryption servers.

2.2 Bilinear Maps and Hardness Assumptions

We use groups $(\mathbb{G}, \mathbb{G}_T)$ of composite order $N = p_1p_2p_3$ endowed with an efficiently computable map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ such that: (1) $e(g^a, h^b) = e(g,h)^{ab}$ for any $(g,h) \in \mathbb{G} \times \mathbb{G}$ and $a, b \in \mathbb{Z}$; (2) if $e(g,h) = 1_{\mathbb{G}_T}$ for each $h \in \mathbb{G}$, then $g = 1_{\mathbb{G}}$. An important property of composite order groups is that pairing two elements of order p_i and p_j, with $i \neq j$, always gives the identity element $1_{\mathbb{G}_T}$.

In the following, for each $i \in \{1,2,3\}$, we denote by $\mathbb{G}_{p_i}$ the subgroup of order p_i. For all distinct $i, j \in \{1,2,3\}$, we call $\mathbb{G}_{p_ip_j}$ the subgroup of order p_ip_j. In this setting, we rely on the following assumptions introduced in [29].

Assumption 1. Given a description of $(\mathbb{G}, \mathbb{G}_T)$ as well as $g \stackrel{R}{\leftarrow} \mathbb{G}_{p_1}, X_3 \stackrel{R}{\leftarrow} \mathbb{G}_{p_3}$ and $T \in \mathbb{G}$, it is infeasible to efficiently decide if $T \in \mathbb{G}_{p_1p_2}$ or $T \in \mathbb{G}_{p_1}$.

Assumption 2. Let $g, X_1 \stackrel{R}{\leftarrow} \mathbb{G}_{p_1}, X_2, Y_2 \stackrel{R}{\leftarrow} \mathbb{G}_{p_2}, Y_3, Z_3 \stackrel{R}{\leftarrow} \mathbb{G}_{p_3}$. Given a description of $(\mathbb{G}, \mathbb{G}_T)$, a set of group elements (g, X_1X_2, Z_3, Y_2Y_3) and T, it is hard to decide if $T \in_R \mathbb{G}_{p_1p_3}$ or $T \in_R \mathbb{G}$.

Assumption 3. Let $g \stackrel{R}{\leftarrow} \mathbb{G}_{p_1}$, $X_2, Y_2, Z_2 \stackrel{R}{\leftarrow} \mathbb{G}_{p_2}$, $X_3 \stackrel{R}{\leftarrow} \mathbb{G}_{p_3}$ and $\alpha, s \stackrel{R}{\leftarrow} \mathbb{Z}_N$. Given a description of $(\mathbb{G}, \mathbb{G}_T)$, group elements $(g, g^\alpha X_2, X_3, g^s Y_2, Z_2)$ and T, it is infeasible to decide if $T = e(g,g)^{\alpha s}$ or $T \in_R \mathbb{G}_T$.

3 A Non-interactive CCA2-Secure Threshold Cryptosystem with Adaptive Corruptions

Our starting point is applying the Canetti-Halevi-Katz [13] transform to a (conceptually equivalent) variant of the Lewko-Waters IBE [29] in the same way as [4]

derives a CCA2-secure threshold cryptosystem from the Boneh-Boyen IBE [3]. We show that composite order groups and the techniques of [29] make it possible to handle adaptive corruptions in a relatively simple way and without having to refresh private key shares after each private key operation.

To this end, we apply a modification to the IBE scheme [29][Section 3]. The latter encrypts M under the identity $\mathsf{ID} \in \mathbb{Z}_N$ as $(M \cdot e(g,g)^{\alpha \cdot s}, g^s, (u^{\mathsf{ID}} \cdot v)^s)$ for a random exponent $s \in \mathbb{Z}_N$ and where the public key is $\big(g, u, v, e(g,g)^{\alpha}\big)$, with $g, u, v \in \mathbb{G}_{p_1}$. We implicitly use an IBE scheme where messages are encrypted as $(M \cdot e(g,h)^{\alpha \cdot s}, g^s, (u^{\mathsf{ID}} \cdot v)^s)$, where $h \neq g$ and $e(g,h)^{\alpha}$ is part of the public key.

Another difference is that, in order to ensure the consistency of these scheme (as defined in section 2.1), the ciphertext validation algorithm has to reject all ciphertexts containing components in the subgroup $\mathbb{G}_{p_3}$.

3.1 Description

In the description hereafter, the verification key of the one-time signature is interpreted as an element of $\mathbb{Z}_N$. In practice, longer keys can be hashed into $\mathbb{Z}_N$ using a collision-resistant hash function.

Setup(λ, t, n)**:** given a security parameter $\lambda \in \mathbb{N}$ and integers $t, n \in \mathsf{poly}(\lambda)$ (with $1 \leq t \leq n$), the algorithm does the following.

1. Choose bilinear groups $(\mathbb{G}, \mathbb{G}_T)$ of order $N = p_1 p_2 p_3$, with $p_1, p_2, p_3 > 2^{\lambda}$.
2. Choose $\alpha \xleftarrow{R} \mathbb{Z}_N$, $g, h, u, v \xleftarrow{R} \mathbb{G}_{p_1}$, $X_{p_3} \xleftarrow{R} \mathbb{G}_{p_3}$ and compute $e(g,h)^{\alpha}$.
3. Choose a strongly unforgeable one-time signature $\Sigma = (\mathcal{G}, \mathcal{S}, \mathcal{V})$.
4. Choose a polynomial $P[X] = \alpha + \alpha_1 X + \cdots + \alpha_{t-1} X^{t-1} \in \mathbb{Z}_N[X]$, for random coefficients $\alpha_1, \ldots, \alpha_{t-1} \xleftarrow{R} \mathbb{Z}_N$. Define the public key to be

$$PK = \Big((\mathbb{G}, \mathbb{G}_T),\ N,\ g,\ e(g,h)^{\alpha},\ u,\ v,\ X_{p_3},\ \Sigma \Big)$$

   and set private key shares $\mathbf{SK} = (SK_1, \ldots, SK_n)$ as $SK_i = h^{P(i)} \cdot Z_{3,i}$, for $i = 1$ to n, with $Z_{3,1}, \ldots, Z_{3,n} \xleftarrow{R} \mathbb{G}_{p_3}$. Verification keys are then set as $\mathbf{VK} = (VK_1, \ldots, VK_n)$ with $VK_i = e(g,h)^{P(i)}$ for $i = 1$ to n.

The public key PK and the verification key $\mathbf{VK}$ are made publicly available while, for each $i \in \{1, \ldots, n\}$, SK_i is given to decryption server i.

Encrypt(PK, m)**:** to encrypt $m \in \mathbb{G}_T$, generate a one-time signature key pair $(\mathsf{SSK}, \mathsf{SVK}) \leftarrow \mathcal{G}(\lambda)$. Choose $s \xleftarrow{R} \mathbb{Z}_N$ and compute

$$C = \big(\mathsf{SVK}, C_0, C_1, C_2, \sigma\big) = \Big(\mathsf{SVK},\ m \cdot e(g,h)^{\alpha \cdot s},\ g^s,\ (u^{\mathsf{SVK}} \cdot v)^s,\ \sigma\Big),$$

where $\sigma = \mathcal{S}(\mathsf{SSK}, (C_0, C_1, C_2))$.

Ciphertext-Verify$\big(PK, C\big)$**:** parse the ciphertext C as $(\mathsf{SVK}, C_0, C_1, C_2, \sigma)$. Return 1 if $\mathcal{V}(\mathsf{SVK}, (C_0, C_1, C_2), \sigma) = 1$, $e(C_j, X_{p_3}) = 1_{\mathbb{G}_T}$ for $j \in \{1, 2\}$ and $e(g, C_2) = e(C_1, u^{\mathsf{SVK}} \cdot v)$. Otherwise, return 0.

Share-Decrypt(i, SK_i, C)**:** Parse C as $\big(\mathsf{SVK}, C_0, C_1, C_2, \sigma\big)$ and SK_i as an element of $\mathbb{G}$. Return $(i, \perp)$ if **Ciphertext-Verify**$\big(PK, C\big) = 0$. Otherwise, choose $r \xleftarrow{R} \mathbb{Z}_N$, $W_3, W_3' \xleftarrow{R} \mathbb{G}_{p_3}$, compute and return $\mu_i = (i, \hat{\mu_i})$, where

$$\hat{\mu_i} = (D_{i,1}, D_{i,2}) = \big(SK_i \cdot (u^{\mathsf{SVK}} \cdot v)^r \cdot W_3,\ g^r \cdot W_3'\big). \tag{1}$$

Share-Verify$\big(PK, C, (i, \hat{\mu_i})\big)$**:** parse C as $(\mathsf{SVK}, C_0, C_1, C_2, \sigma)$. If $\hat{\mu_i} = \perp$ or $\hat{\mu_i} \notin \mathbb{G}^2$, return 0. Otherwise, parse $\hat{\mu_i}$ as a pair $(D_{i,1}, D_{i,2}) \in \mathbb{G}^2$ and return 1 if $e(D_{i,1}, g) = VK_i \cdot e(u^{\mathsf{SVK}} \cdot v, D_{i,2})$. In any other situation, return 0.

Combine$(PK, C, \{(i, \hat{\mu_i})\}_{i \in S})$**:** for each $i \in S$, parse the share $\hat{\mu_i}$ as $(D_{i,1}, D_{i,2})$ and return $\perp$ if **Share-Verify**$\big(PK, C, (i, \hat{\mu_i})\big) = 0$. Otherwise, compute $(D_1, D_2) = \big(\prod_{i \in S} D_{i,1}^{\Delta_{i,S}(0)}, \prod_{i \in S} D_{i,2}^{\Delta_{i,S}(0)}\big)$, which equals

$$(D_1, D_2) = \big(h^{\alpha} \cdot (u^{\mathsf{SVK}} \cdot v)^{\tilde{r}} \cdot \tilde{W}_3,\ g^{\tilde{r}} \cdot \tilde{W}_3'\big),$$

for some $\tilde{W}_3, \tilde{W}_3' \in \mathbb{G}_{p_3}$ and $\tilde{r} \in \mathbb{Z}_{p_1}$. Using (D_1, D_2), compute and output the plaintext $m = C_0 \cdot e(C_1, D_1)^{-1} \cdot e(C_2, D_2)$.

As far as efficiency goes, the ciphertext-validity check can be optimized by choosing $\omega_1, \omega_2 \xleftarrow{R} \mathbb{Z}_N$ and checking that $e(g \cdot X_{p_3}^{\omega_1}, C_2) = e(C_1, (u^{\mathsf{SVK}} \cdot v) \cdot X_{p_3}^{\omega_2})$, which rejects ill-formed ciphertexts with overwhelming probability and saves two pairing evaluations. Similar batch verification techniques apply to simultaneously test t or more decryption shares using only two pairing evaluations[2].

We observe that, as in [4], decryption shares can be seen as signature shares (for a message consisting of the verification key SVK) calculated by decryption servers. In the full version, we show that the underlying threshold signature is secure against chosen-message attacks in the adaptive corruption scenario.

3.2 Security

The security proof departs from approaches that were previously used in threshold cryptography in that we do not construct an adversary against the centralized version of the scheme out of a CCA2 adversary against its threshold implementation. Instead, we directly prove the security of the latter using the dual encryption paradigm [37,29].

Our proof proceeds with a sequence of games and uses semi-functional ciphertexts as in [29], and decryption shares. Still, there are two differences. First, two kinds of semi-functional ciphertexts (that differ in the presence of a component of order p_2 in the target group $\mathbb{G}_T$) have to be involved. The second difference is that we need to introduce semi-functional private key shares at some step of the proof and argue that they cannot be distinguished from real key shares. The proof takes advantage of the fact that, at each step of the sequence, the simulator knows either the $\mathbb{G}_{p_1}$ components of private key shares $\{h^{P(i)}\}_{i=1}^n$ or a "blinded" version $\{h^{P(i)} \cdot Z_{2,i}\}_{i=1}^n$ of those shares, for some $Z_{2,i} \in_R \mathbb{G}_{p_2}$, which suffices to consistently answer adaptive corruption queries.

[2] Namely, t shares $\{\mu_i = (D_{i,1}, D_{i,2})\}_{i=1}^t$ can be batch-verified by drawing $\omega_1, \ldots, \omega_t \xleftarrow{R} \mathbb{Z}_N$ and testing if $e\big(g, \prod_{i=1}^t D_{i,1}^{\omega_i}\big) = \prod_{i=1}^t VK_i^{\omega_i} \cdot e\big(u^{\mathsf{SVK}} \cdot v, \prod_{i=1}^t D_{i,2}^{\omega_i}\big)$.

Theorem 1. *The scheme is IND-CCA2 against adaptive corruptions assuming that Assumption 1, Assumption 2 and Assumption 3 all hold and that Σ is a strongly unforgeable*[3] *one-time signature.*

Proof. The proof proceeds using a sequence of games including steps similar to [29] and additional steps. As in [37,29], the proof makes use of semi-functional ciphertexts and decryption shares (which are actually private keys in [29]). In addition, we also have to consider semi-functional private key shares. Another difference is that we need two kinds of semi-functional ciphertexts.

- Semi-functional ciphertexts of Type I are generated from a normal ciphertext (C'_0, C'_1, C'_2) and some $g_2 \in \mathbb{G}_{p_2}$, by choosing random $\tau, z_c \xleftarrow{R} \mathbb{Z}_N$ and setting

$$C_0 = C'_0, \qquad C_1 = C'_1 \cdot g_2^{\tau}, \qquad C_2 = C'_2 \cdot g_2^{\tau z_c}.$$

- Semi-functional ciphertexts of Type II are generated from a normal ciphertext (C'_0, C'_1, C'_2) by choosing random $\tau, z_c, \theta \xleftarrow{R} \mathbb{Z}_N$ and setting

$$C_0 = C'_0 \cdot e(g_2, g_2)^{\theta}, \qquad C_1 = C'_1 \cdot g_2^{\tau}, \qquad C_2 = C'_2 \cdot g_2^{\tau z_c}.$$

- Semi-functional decryption shares are obtained from a normal decryption share $(D'_{i,1}, D'_{i,2})$ by picking $\gamma, z_k \xleftarrow{R} \mathbb{Z}_N$, $W_3, W'_3 \xleftarrow{R} \mathbb{G}_{p_3}$ and setting

$$D_{i,1} = D'_{i,1} \cdot g_2^{\gamma z_k} \cdot W_3, \qquad D_{i,2} = D'_{i,2} \cdot g_2^{\gamma} \cdot W'_3.$$

- Semi-functional private key shares $\{SK_i\}_{i=1}^n$ are obtained from normal shares $\{SK'_i\}_{i=1}^n$ by setting $SK_i = SK'_i \cdot Z_{2,i}$, where $Z_{2,i} \xleftarrow{R} \mathbb{G}_{p_2}$, for $i = 1$ to n.

The proof considers a sequence of $q + 6$ games. It starts with the real game Game_{real} followed by $\mathsf{Game}_{restricted}$, $\mathsf{Game}^*_{restricted}$ $\mathsf{Game}_0, \mathsf{Game}_1, \ldots, \mathsf{Game}_q$ and finally Game^*_q and Game_{final}.

$\mathsf{Game}_{restricted}$**:** is identical to Game_{real} with the difference that the challenger $\mathcal{B}$ rejects all post-challenge decryption queries $(\mathsf{SVK}, C_0, C_1, C_2, \sigma)$ for which $\mathsf{SVK} = \mathsf{SVK}^\star$, where $\mathsf{SVK}^\star$ denotes the one-time verification key included in the challenge ciphertext.

$\mathsf{Game}^*_{restricted}$**:** is identical to $\mathsf{Game}_{restricted}$ with the difference that the adversary $\mathcal{A}$ is not allowed to make decryption queries $(\mathsf{SVK}, C_0, C_1, C_2, \sigma)$ for which $\mathsf{SVK} = \mathsf{SVK}^\star \bmod p_2$.

Game_0**:** is identical to $\mathsf{Game}^*_{restricted}$ but the normal challenge ciphertext is replaced by a semi-functional ciphertext of Type I.

Game_k $(1 \leq k \leq q)$**:** in this game, the challenge ciphertext is a semi-functional ciphertext of Type I and the challenger $\mathcal{B}$ answers the first k decryption queries by returning semi-functional decryption shares. As for the last $q - k$ decryption queries, they are answered using normal decryption shares.

[3] Strong unforgeability refers to the infeasibility, after having obtained a message-signature pair (M, σ), of computing a new pair $(M^\star, \sigma^\star) \neq (M, \sigma)$.

Game_q^***:** is identical to Game_q with the following two differences.

- All private key shares are made semi-functional and thus contain a random $\mathbb{G}_{p_2}$ component.
- The Type I semi-functional challenge ciphertext is traded for a semi-functional ciphertext of Type II.

Game_{final}**:** is as Game_q^* but the Type II semi-functional challenge ciphertext is replaced by a semi-functional encryption of a random plaintext (instead of M_β). In this game, $\mathcal{A}$ has no information on the challenger's bit $\beta \in \{0,1\}$ and cannot guess it with better probability than 1/2.

As in [29], when a semi-functional decryption share is used (in combination with $t-1$ normal decryption shares) to decrypt a semi-functional ciphertext, decryption only works when $z_k = z_c$, in which case the decryption share is called *nominally* semi-functional. For each $k \in \{1,\ldots,q\}$, the transitions between Game_{k-1} and Game_k is done in such a way that the distinguisher cannot directly decide (*i.e.*, without interacting with $\mathcal{A}$) whether the k^{th} decryption share is normal or semi-functional by generating this share for the challenge verification key $\mathsf{SVK}^\star$. Indeed, in such an attempt, the generated decryption share is necessarily either normal or nominally semi-functional, so that decryption succeeds either way.

Moreover, during the transition between Game_q and Game_q^*, we have to make sure that the distinguisher cannot bypass its interaction with the adversary and try to distinguish the two games by itself either. Should it attempt to decrypt the challenge ciphertext using the private key shares, the transition is organized in such a way that decryption succeeds regardless of whether the private key shares (resp. the challenge ciphertext) are normal or semi-functional (resp. semi-functional of Type I or II).

The proof is completed by lemma 1 to 6, which show that all games are computationally indistinguishable as long as the one-time signature is strongly unforgeable and Assumptions 1, 2, 3 hold. □

Lemma 1. *If the one-time signature Σ is strongly unforgeable,* Game_{real} *and* $\mathsf{Game}_{restricted}$ *are indistinguishable.*

Proof. The proof uses the classical argument saying that the only way for the adversary to create a legal decryption query $(\mathsf{SVK}^\star, C_0, C_1, C_2, \sigma)$ after the challenge phase is to break the strong unforgeability of Σ. Moreover, the challenge one-time verification key can be defined at the very beginning of the game. Hence, computing a valid pre-challenge decryption query involving $\mathsf{SVK}^\star$ would require the adversary to compute a valid signature without having seen a single signature (or even the verification key) and *a fortiori* break the security of Σ. □

Lemma 2. *Provided Assumption 1 and Assumption 2 both hold,* $\mathsf{Game}_{restricted}$ *and* $\mathsf{Game}^*_{restricted}$ *are indistinguishable.*

Proof. The proof is identical to the one of lemma 5 in [29]. Namely, the only situation where the two games are distinguishable is when the adversary $\mathcal{A}$ manages to come up with a ciphertext for which $\mathsf{SVK}^\star \neq \mathsf{SVK}$ but $\mathsf{SVK}^\star = \mathsf{SVK} \bmod p_2$.

In this case, the challenger $\mathcal{B}$ can compute $\gcd(\mathsf{SVK} - \mathsf{SVK}^\star, N)$, which is necessarily a non-trivial factor of N. Depending on which factor is found, $\mathcal{B}$ can break either Assumption 1 or Assumption 2. □

The proofs of lemma 3 and 4 proceed exactly as in [29] and we give them in the full version of the paper.

Lemma 3. *Under Assumption 1, no PPT adversary can distinguish* $\mathsf{Game}^*_{restricted}$ *and* Game_0.

Lemma 4. *Under Assumption 2, no PPT adversary can distinguish* Game_k *from* Game_{k-1} *for* $1 \leq k \leq q$.

In comparison with the security proof of [29], the novelty is the transition from Game_q to Game^*_q, which is addressed in lemma 5. This transition allows turning all private key shares into semi-functional shares in one step.

Lemma 5. *Under Assumption 2,* Game_q *and* Game^*_q *are indistinguishable.*

Proof. Towards a contradiction, we assume that a PPT adversary $\mathcal{A}$ can tell apart Game_q and Game^*_q. We construct an algorithm $\mathcal{B}$ that, given elements $(g, X_3, X_1X_2, Y_2Y_3, T)$, decides if $T \in_R \mathbb{G}_{p_1p_3}$ or $T \in_R \mathbb{G}$.

Algorithm $\mathcal{B}$ prepares PK by setting $X_{p_3} = X_3$, $u = g^a$ and $v = g^b$ with $a, b \xleftarrow{R} \mathbb{Z}_N$. It also picks $\alpha \xleftarrow{R} \mathbb{Z}_N$ and defines $e(g,h)^\alpha = e(g,T)^\alpha$. In addition, $\mathcal{B}$ chooses a random polynomial $P[X]$ of degree $t-1$ such that $P(0) = \alpha$. To prepare $\mathbf{SK} = (SK_1, \ldots, SK_n)$ and $\mathbf{VK} = (VK_1, \ldots, VK_n)$, $\mathcal{B}$ sets $SK_i = T^{P(i)} \cdot Z_{3,i}$, with $Z_{3,i} \xleftarrow{R} \mathbb{G}_{p_3}$, and $VK_i = e(g, SK_i)$ for $i = 1$ to n.

In the challenge phase, $\mathcal{A}$ outputs M_0, M_1 and $\mathcal{B}$ flips a coin $\beta \xleftarrow{R} \{0,1\}$. It generates $(\mathsf{SSK}^\star, \mathsf{SVK}^\star) \leftarrow \mathcal{G}(\lambda)$ and computes

$$C_0^\star = M_\beta \cdot e(X_1X_2, T)^\alpha, \qquad C_1^\star = X_1X_2, \qquad C_2^\star = (X_1X_2)^{a \cdot \mathsf{SVK}^\star + b} \tag{2}$$

In order to generate a decryption share on behalf of decryption server i for a ciphertext $(\mathsf{SVK}, C_0, C_1, C_2, \sigma)$, $\mathcal{B}$ chooses $r, w, w' \xleftarrow{R} \mathbb{Z}_N$ and generates a semi-functional decryption share

$$(D_{i,1}, D_{i,2}) = (SK_i \cdot (u^{\mathsf{SVK}} \cdot v)^r \cdot (Y_2Y_3)^w,\ g^r \cdot (Y_2Y_3)^{w'}). \tag{3}$$

Whenever $\mathcal{A}$ decides to corrupt decryption server i, $\mathcal{B}$ simply reveals SK_i.

We note that, in the situation where $T \in \mathbb{G}_{p_1p_3}$, $\mathcal{B}$ is clearly playing Game_q. Now, let us consider what happens when $T \in_R \mathbb{G}$. In this case, T can be written as $T = g^{\gamma_1} g_2^{\gamma_2} g_3^{\gamma_3}$ for some random $\gamma_1 \in \mathbb{Z}_{p_1}$, $\gamma_2 \in \mathbb{Z}_{p_2}$, $\gamma_3 \in \mathbb{Z}_{p_3}$ and h is implicitly set as $h = g^{\gamma_1}$. From the simulator's standpoint, the $\mathbb{G}_{p_2}$ components of private key shares $\{SK_i\}_{i=1}^n$ are not independent since a polynomial of degree $t-1$ goes through them in the exponent: for each index $i \in \{1, \ldots, n\}$, we indeed have $SK_i = g^{\gamma_1 \cdot P(i)} \cdot g_2^{\gamma_2 \cdot P(i)} \cdot \tilde{Z}_{3,i}$, for some $\tilde{Z}_{3,i} \in_R \mathbb{G}_{p_3}$. In addition, if we write $X_1X_2 = g^s \cdot g_2^\tau$, for some $s \in \mathbb{Z}_{p_1}$, $\tau \in \mathbb{Z}_{p_2}$, the components

$$(C_0^\star, C_1^\star, C_2^\star) = \left(M_\beta \cdot e(g,h)^{\alpha \cdot s} \cdot e(g_2, g_2)^{\gamma_2 \cdot \tau \cdot \alpha},\ g^s \cdot g_2^\tau,\ (u^{\mathsf{SVK}^\star} \cdot v)^s \cdot g_2^{\tau z_c}\right)$$

of the challenge ciphertext information-theoretically reveal $e(g_2, g_2)^{\gamma_2 \cdot \tau \cdot P(0)}$.

However, the public key does not reveal anything about $\alpha \bmod p_2$ and the distribution of $\mathbf{VK}$ is uncorrelated to $\{P(i) \bmod p_2\}_{i=1}^n$. Moreover, as decryption shares are generated as per (3), they perfectly hide $P(i) \bmod p_2$ in the first term SK_i of the product $D_{i,1}$. Hence, since the adversary $\mathcal{A}$ cannot obtain more than $t-1$ private key shares throughout the game, the correlation between the $\mathbb{G}_{p_2}$ components of $\{SK_i\}_{i=1}^n$ is information-theoretically hidden to $\mathcal{A}$. Indeed, given that $P[X] \in \mathbb{Z}_N[X]$ is chosen as a random polynomial of degree $t-1$, its evaluations $P(i) \bmod p_2$ are t-wise independent. In other words, for any $(t-1)$-subset $\mathcal{C} \subset \{1, \ldots, n\}$ chosen by $\mathcal{A}$, the values $\{g_2^{\gamma_2 \cdot P(i)}\}_{i \in \mathcal{C} \cup \{0\}}$ are statistically indistinguishable from a set of t random elements of $\mathbb{G}_{p_2}$. This means that, from $\mathcal{A}$'s view, $(C_0^\star, C_1^\star, C_2^\star)$ and $\{SK_i\}_{i \in \mathcal{C}}$ look like a Type II semi-functional ciphertext and a set of $t-1$ semi-functional private key shares, respectively. We conclude that, if $T \in_R \mathbb{G}$, the simulator $\mathcal{B}$ is actually playing Game_q^* with $\mathcal{A}$. □

In the proof of lemma 5, we note that $\mathcal{B}$ cannot distinguish $T \in_R \mathbb{G}_{p_1 p_3}$ from $T \in_R \mathbb{G}$ by itself: if $\mathcal{B}$ tries to decrypt the challenge ciphertext (given by (2)) using the decryption shares $\{SK_i\}_{i=1}^n$, decryption recovers M_β in either case.

Lemma 6. *Under Assumption 3, no PPT adversary can distinguish* Game_q^* *from* Game_{final} *(the proof is deferred to the full version of the paper).*

Unlike [35,4], where consistency holds statistically, we demonstrate consistency in the computational sense and prove the next result in the full version.

Theorem 2. *The scheme provides consistency if Assumption 1 holds.*

References

1. Almansa, J.F., Damgård, I.B., Nielsen, J.B.: Simplified Threshold RSA with Adaptive and Proactive Security. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 593–611. Springer, Heidelberg (2006)
2. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: ACM CCS (1993)
3. Boneh, D., Boyen, X.: Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004)
4. Boneh, D., Boyen, X., Halevi, S.: Chosen Ciphertext Secure Public Key Threshold Encryption Without Random Oracles. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 226–243. Springer, Heidelberg (2006)
5. Boneh, D., Franklin, M.: Identity-Based Encryption from the Weil Pairing. SIAM J. of Computing 32(3), 586–615 (2003); Earlier version in Crypto 2001
6. Boneh, D., Franklin, M.: Efficient Generation of Shared RSA Keys. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 425–439. Springer, Heidelberg (1997)

7. Boyd, C.: Digital Multisignatures. In: Beker, H.J., Piper, F.C. (eds.) Cryptography and Coding, pp. 241–246. Oxford University Press, Oxford (1989)
8. Boyen, X., Mei, Q., Waters, B.: Direct Chosen Ciphertext Security from Identity-Based Techniques. In: ACM CCS 2005 (2005)
9. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. Journal of the ACM 51(4), 557–594 (2004); Earlier version in STOC 1998 (1998)
10. Canetti, R., Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Adaptive Security for Threshold Cryptosystems. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, p. 98. Springer, Heidelberg (1999)
11. Canetti, R., Goldwasser, S.: An Efficient Threshold Public Key Cryptosystem Secure Against Adaptive Chosen Ciphertext Attack. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, p. 90. Springer, Heidelberg (1999)
12. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, p. 13. Springer, Heidelberg (1998)
13. Canetti, R., Halevi, S., Katz, J.: Chosen-Ciphertext Security from Identity-Based Encryption. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 207–222. Springer, Heidelberg (2004)
14. Cramer, R., Damgård, I.B., Dziembowski, S., Hirt, M., Rabin, T.: Efficient Multiparty Computations Secure Against an Adaptive Adversary. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, p. 311. Springer, Heidelberg (1999)
15. Daza, V., Herranz, J., Morillo, P., Ràfols, C.: CCA2-Secure Threshold Broadcast Encryption with Shorter Ciphertexts. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 35–50. Springer, Heidelberg (2007)
16. Delerablée, C., Pointcheval, D.: Dynamic Threshold Public-Key Encryption. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 317–334. Springer, Heidelberg (2008)
17. Desmedt, Y.: Society and Group Oriented Cryptography: A New Concept. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 120–127. Springer, Heidelberg (1988)
18. Desmedt, Y.G., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 307–315. Springer, Heidelberg (1990)
19. Dodis, Y., Katz, J.: Chosen-Ciphertext Security of Multiple Encryption. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 188–209. Springer, Heidelberg (2005)
20. El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985)
21. Goldwasser, S., Micali, S., Rivest, R.: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. SIAM J. Comput. 17(2), 281–308 (1988)
22. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, p. 295. Springer, Heidelberg (1999)
23. Fouque, P.-A., Pointcheval, D.: Threshold Cryptosystems Secure against Chosen-Ciphertext Attacks. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, p. 351. Springer, Heidelberg (2001)
24. Frankel, Y., MacKenzie, P.D., Yung, M.: Adaptively-Secure Distributed Public-Key Systems. In: Nešetřil, J. (ed.) ESA 1999. LNCS, vol. 1643, pp. 4–27. Springer, Heidelberg (1999)
25. Frankel, Y., MacKenzie, P.D., Yung, M.: Adaptively-Secure Optimal-Resilience Proactive RSA. In: Lam, K.-Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT 1999. LNCS, vol. 1716, pp. 180–195. Springer, Heidelberg (1999)

26. Jarecki, S., Lysyanskaya, A.: Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures (Extended Abstract). In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, p. 221. Springer, Heidelberg (2000)
27. Kiltz, E.: Chosen-ciphertext security from tag-based encryption. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 581–600. Springer, Heidelberg (2006)
28. Lewko, A., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 62–91. Springer, Heidelberg (2010)
29. Lewko, A., Waters, B.: New Techniques for Dual System Encryption and Fully Secure HIBE with Short Ciphertexts. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 455–479. Springer, Heidelberg (2010)
30. Lysyanskaya, A., Peikert, C.: Adaptive Security in the Threshold Setting: From Cryptosystems to Signature Schemes. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, p. 331. Springer, Heidelberg (2001)
31. MacKenzie, P.: An Efficient Two-Party Public Key Cryptosystem Secure against Adaptive Chosen Ciphertext Attack. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567. Springer, Heidelberg (2002)
32. Ostrovsky, R., Yung, M.: How to Withstand Mobile Virus Attacks. In: 10th ACM Symp. on Principles of Distributed Computing, PODC 1991 (1991)
33. Qin, B., Wu, Q., Zhang, L., Domingo-Ferrer, J.: Threshold Public-Key Encryption with Adaptive Security and Short Ciphertexts. In: Soriano, M., Qing, S., López, J. (eds.) ICICS 2010. LNCS, vol. 6476, pp. 62–76. Springer, Heidelberg (2010)
34. Shamir, A.: Identity-Based Cryptosystems and Signature Schemes. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985)
35. Shoup, V., Gennaro, R.: Securing Threshold Cryptosystems against Chosen Ciphertext Attack. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 1–16. Springer, Heidelberg (1998)
36. Shoup, V.: Practical Threshold Signatures. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, p. 207. Springer, Heidelberg (2000)
37. Waters, B.: Dual System Encryption: Realizing Fully Secure IBE and HIBE under Simple Assumptions. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 619–636. Springer, Heidelberg (2009)

Content Search through Comparisons

Amin Karbasi[1], Stratis Ioannidis[2], and Laurent Massoulié[3]

[1] Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland
[2] Technicolor, Palo Alto, USA
[3] Technicolor, Paris, France

Abstract. We study the problem of navigating through a database of similar objects using comparisons under heterogeneous demand, a problem closely related to small-world network design. We show that, under heterogeneous demand, the small-world network design problem is NP-hard. Given the above negative result, we propose a novel mechanism for small-world network design and provide an upper bound on its performance under heterogeneous demand. The above mechanism has a natural equivalent in the context of content search through comparisons, again under heterogeneous demand; we use this to establish both upper and lower bounds on content search through comparisons.

1 Introduction

The problem we study in this paper is content search through comparisons. In short, a user searching for a target object navigates through a database in the following manner. The user is asked to select the object most similar to her target from small list of objects. A new object list is then presented to the user based on her earlier selection. This process is repeated until the target is included in the list presented, at which point the search terminates.

Searching through comparisons is typical example of *exploratory search* [14], the need for which arises when users are unable to state and submit explicit queries to the database. Exploratory search has several important real-life applications. An often-cited example [13, 12] is navigating through a database of pictures of humans in which subjects are photographed under diverse uncontrolled conditions. For example, the pictures may be taken outdoors, from different angles or distances, while the subjects assume different poses, are partially obscured, *etc.* Automated methods may fail to extract meaningful features from such photos, so the database cannot be queried in the traditional fashion. On the other hand, a human searching for a particular person can easily select from a list of pictures the subject most similar to the person she has in mind.

Users may also be unable to state queries because, *e.g.*, the are unfamiliar with the search domain, or do not have a clear target in mind. For example, a novice classical music listener may not be able to express that she is, *e.g.*, looking for a fugue or a sonata. She might however identify among samples of different musical pieces the closest to the one she has in mind. Alternatively, a user surfing the web may not know a priori which post she wishes to read;

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 601–612, 2011.

presenting a list of blog posts and letting the surfer identify which one she likes best can steer her in the right direction.

In all the above applications, the problem of content search through comparisons amounts to determining which objects to present to the user in order to find the target object as quickly as possible. Formally, the behavior of a human user can be modeled by a so-called *comparison oracle* [5]: given a target and a choice between two objects, the oracle outputs the one closest to the target. The goal is thus to find a sequence of proposed pairs of objects that leads to the target object with as few oracle queries as possible. This problem was introduced in [5] and has recently received considerable attention [11, 12, 13].

Content search through comparisons is also naturally related to the following problem: given a graph embedded in a metric space, how should one augment this graph by adding edges in order to minimize the expected cost of greedy forwarding over this graph? This is known as the *small-world network design* problem [4, 3] and has a variety of applications as, *e.g.*, in network routing. In this paper, we consider both problems under the scenario of *heterogeneous demand*. This is very interesting in practice: objects in a database are indeed unlikely to be requested with the same frequency. Our contributions are as follows:

- We show that the small-world network design problem under general heterogeneous demand is NP-hard. Given earlier work on this problem under homogeneous demand [3, 4], this result is interesting in its own right.
- We propose a novel mechanism for edge addition in the small-world design problem, and provide an upper bound on its performance.
- The above mechanism has a natural equivalent in the context of content search through comparisons, and we provide a matching upper bound for the performance of this mechanism.
- Finally, we also establish a lower bound on any mechanism solving the content search through comparisons problem.

To the best of our knowledge, we are the first to study the above two problems in a setting of heterogeneous demand. Our analysis is intuitively appealing because our upper and lower bounds relate the cost of content search to two important properties of the demand distribution, namely its *entropy* and its *doubling constant*. We thus provide performance guarantees in terms of the *bias* of the distribution of targets, captured by the entropy, as well as the *topology* of their embedding, captured by the doubling constant.

The remainder of this paper is organized as follows. In Section 2 we provide an overview of the related work in this area. In Sections 3 and 4 we introduce our notation and formally state the two problems that are the focus of this work, namely content search through comparisons and small-world network design. We present our main results in Section 5 and our conclusions in Section 6.

2 Related Work

Content search through comparisons is a special case of nearest neighbour search (NNS) [1, 6], where it is typical to assume that database objects are embedded

in a metric space with a small intrinsic dimension. Krauthgamer and Lee [10] introduce navigating nets, a data structure for NNS in doubling metric spaces. Clarkson [1] considers a similar structure for objects embedded in a space satisfying a sphere-packing property, while Karger and Ruhl [8] study NNS under growth-restricted metrics. All three assumptions have formal connections to the doubling constant we consider in this paper. However, in these works, the underlying metric space is fully observable by the search mechanism, and the demand over target objects is homogeneous. Our work assumes access only to a comparison oracle while also dealing with heterogeneous demand.

NNS with access to a comparison oracle was first introduced by Lifshits *et al.* [5], and further explored by Lifshits and Zhang [11] and Tshopp and Diggavi [12, 13]. In contrast to [8, 10, 1], the above authors do not assume that objects are necessarily embedded in a metric space; instead, they only require that a comparison oracle can rank any two objects in terms of their similarity to a given target. To provide performance guarantees on the search cost, Lifshits *et al.* introduce a *disorder constant* [5], capturing the degree to which object rankings violate the triangle inequality. This disorder constant plays roughly the same role in their analysis as the doubling constant does in ours. Nevertheless, these works also assume homogeneous demand. Our work introduces the notion of heterogeneity while assuming that a metric embedding exists.

Small-world networks (also called navigable networks) have received a lot of attention since Kleinberg's seminal paper [9]. Our work is closest to Fraigneaud *et al.* [4], [3], who identify conditions under which graphs embedded in a doubling metric space are navigable. Again, our approach to small-world network design differs by considering heterogeneous demand, an aspect absent from earlier work.

3 Definitions and Notation

Comparison Oracle. Consider a set of objects $\mathcal{N}$, where $|\mathcal{N}| = n$, and a metric space $(\mathcal{M},d)$, where $d(x, y)$ denotes the distance between $x, y \in \mathcal{M}$. Assume that objects in $\mathcal{N}$ are embedded in $(\mathcal{M},d)$, *i.e.*, there exists a 1-to-1 mapping from $\mathcal{N}$ to a subset of $\mathcal{M}$. The objects in $\mathcal{N}$ may represent, for example, pictures in a database. The metric embedding is a mapping from the pictures to a set of attributes (*e.g.*, the person's age, her eye color, *etc.*). The distance d then represents how "similar" objects are w.r.t. these attributes. In what follows, we abuse notation and write $\mathcal{N} \subseteq \mathcal{M}$, keeping in mind that database objects (the pictures) are in fact distinct from their embedding (their attributes).

Given an object $z \in \mathcal{N}$, we write $x \preccurlyeq_z y$ if $d(x, z) \leq d(y, z)$, ordering thus objects according to their distance from z. Moreover, we write $x \sim_z y$ if $d(x, z) = d(y, z)$ and $x \prec_z y$ if $x \preccurlyeq_z y$ but not $x \sim_z y$. For a non-empty $A \subseteq \mathcal{N}$, let $\min_{\preccurlyeq_z} A$ be the set of objects in A closest to z, *i.e.*, $w \in \min_{\preccurlyeq_z} A \subseteq A$ if $w \preccurlyeq_z v$ for all $v \in A$.

A *comparison oracle* [5] is an oracle that, given two objects x, y and a target t, returns the closest object to t. More formally,

$$\text{Oracle}(x, y, t) = \begin{cases} x & \text{if } x \prec_t y, \\ y & \text{if } x \succ_t y, \\ x \text{ or } y & \text{if } x \sim_t y. \end{cases} \tag{1}$$

This oracle "models" human users: a user interested in locating, *e.g.*, a target picture t within the database, can compare two pictures with respect to their similarity to this target but cannot associate a numerical value to this similarity. When the two pictures are equidistant from t the user's decision is arbitrary.

Entropy and Doubling Constant. For any ordered pair $(s,t) \in \mathcal{N} \times \mathcal{N}$, we call s the *source* and t the *target* of the pair. We consider a probability distribution λ over all ordered pairs of objects in $\mathcal{N}$ which we call the *demand*. We refer to the marginal distributions $\nu(s) = \sum_t \lambda(s,t)$ and $\mu(t) = \sum_s \lambda(s,t)$, as the *source* and *target* distributions, respectively. Moreover, we refer to the support of the target distribution $\mathcal{T} = \mathsf{supp}(\mu) = \{x \in \mathcal{N} : \text{ s.t. } \mu(x) > 0\}$ as the *target set* of the demand.

Let σ be a probability distribution over $\mathcal{N}$. We define the *entropy* and *max-entropy* of σ, respectively, as

$$H(\sigma) = \sum_{x \in \mathsf{supp}(\sigma)} \sigma(x) \log \sigma^{-1}(x), \qquad H_{\max}(\sigma) = \max_{x \in \mathsf{supp}(\sigma)} \log \sigma^{-1}(x). \tag{2}$$

The entropy has strong connections with content search. More specifically, suppose that we have access to a so-called *membership oracle* [2] that answers queries of the following form: "Given a target t and a subset $A \subseteq \mathcal{N}$, does t belong to A?"Let t be a random target selected with distribution μ. To identify t one needs to submit at least $H(\mu)$ queries, in expectation, to a membership oracle, and there exists an algorithm (Huffman coding) that identifies t with only $H(\mu)+1$ queries, in expectation (see, *e.g.*, [2]). In the worst case, which occurs when the target is the least frequently selected object, the algorithm requires $H_{\max}(\mu)+1$ queries to identify t. Our work identifies similar bounds assuming that one only has access to a comparison oracle, as defined in (1). Not surprisingly, the entropy $H(\mu)$ also shows up in our performance bounds (Theorems 3 and 4).

For $x \in \mathcal{N}$, we denote by $B_x(r) = \{y \in \mathcal{M} : d(x,y) \leq r\}$ the closed ball of radius $r \geq 0$ around x. Given a probability distribution σ over $\mathcal{N}$ and a set $A \subset \mathcal{N}$ let $\sigma(A) = \sum_{x \in A} \sigma(x)$. We define the *doubling constant* $c(\sigma)$ to be the minimum $c > 0$ for which $\sigma(B_x(2r)) \leq c \cdot \sigma(B_x(r))$, for any $x \in \mathsf{supp}(\sigma)$ and any $r \geq 0$. As we will see, search trough comparisons depends not only on the entropy $H(\mu)$ but also on the topology of μ, as captured by $c(\mu)$.

4 Problem Statement

We now formally define the two problems we study. The first is *content search through comparisons* and the second is the *small-world network design* problem.

4.1 Content Search Through Comparisons

Consider the object set $\mathcal{N}$. Although its embedding in $(\mathcal{M}, d)$ exists, we are constrained by not being able to directly compute object distances; instead, we

only have access to a comparison oracle. In particular, we define *greedy content search* as follows. Let t be a target and s an object serving as a starting point. The greedy content search algorithm proposes an object w and asks the oracle to select among s and w the object closest to t, *i.e.*, it evokes $\text{Oracle}(s, w, t)$. This is repeated until the oracle returns something other than s, say w'. If $w' \neq t$, the algorithm repeats these steps, now from w'. If $w' = t$, the search terminates.

Formally, for $k = 1, 2, \ldots$, let x_k, y_k be the k-th pair of objects submitted to the oracle: x_k is the *current object*, which greedy content search is trying to improve upon, and y_k is the *proposed object*, submitted to the oracle for comparison with x_k. Let $o_k = \text{Oracle}(x_k, y_k, t) \in \{x_k, y_k\}$ be the oracle's response, and define the *history* of the search up to and including the k-th access as $\mathcal{H}_k = \{(x_i, y_i, o_i)\}_{i=1}^{k}$.

The source object is always one of the first two objects submitted to the oracle, *i.e.*, $x_1 = s$. Moreover, $x_{k+1} = o_k$, *i.e.*, the current object is always the closest to the target so far. The selection of the proposed object y_{k+1} is determined by the history $\mathcal{H}_k$ and the object x_k. In particular, given $\mathcal{H}_k$ and the current object x_k there exists a mapping $(\mathcal{H}_k, x_k) \mapsto \mathcal{F}(\mathcal{H}_k, x_k) \in \mathcal{N}$ such that $y_{k+1} = \mathcal{F}(\mathcal{H}_k, x_k)$, where here we take $x_0 = s \in \mathcal{N}$ (the source/starting object) and $\mathcal{H}_0 = \emptyset$ (*i.e.*, before any comparison takes place, there is no history).

We call the mapping $\mathcal{F}$ the *selection policy* of the greedy content search. In general, we allow the selection policy to be randomized; in this case, the object returned by $\mathcal{F}(\mathcal{H}_k, x_k)$ is a random variable, whose distribution $\Pr(\mathcal{F}(\mathcal{H}_k, x_k) = w)$ for $w \in \mathcal{N}$ is fully determined by $(\mathcal{H}_k, x_k)$. Observe that $\mathcal{F}$ depends on the target t only indirectly, through $\mathcal{H}_k$ and x_k; this is because t is only "revealed" when the search terminates. We say that a selection policy is *memoryless* if it depends on x_k but not on the history $\mathcal{H}_k$.

Our goal is to select an $\mathcal{F}$ that minimizes the number of accesses to the oracle. In particular, given a source object s, a target t and a selection policy $\mathcal{F}$, we define the search cost $C_{\mathcal{F}}(s, t) = \inf\{k : y_k = t\}$ to be the number of proposals to the oracle until t is found. This is a random variable, as $\mathcal{F}$ is randomized; let $\mathbb{E}[C_{\mathcal{F}}(s, t)]$ be its expectation. We thus define the following problem.

> Content Search Through Comparisons (CSTC): Given an embedding of $\mathcal{N}$ into $(\mathcal{M}, d)$ and a demand distribution $\lambda(s, t)$, select $\mathcal{F}$ that minimizes the expected search cost $\bar{C}_{\mathcal{F}} = \sum_{(s,t)\in\mathcal{N}\times\mathcal{N}} \lambda(s, t)\mathbb{E}[C_{\mathcal{F}}(s, t)]$.

Note that, as $\mathcal{F}$ is randomized, the free variable in the above optimization problem is the distribution $\Pr(\mathcal{F}(\mathcal{H}_k, x_k) = w)$.

4.2 Small-World Network Design

In the small-world network design problem the objects in $\mathcal{N}$, embedded in $(\mathcal{M}, d)$, are connected to each other. The network formed by such connections is represented by a directed graph $G(\mathcal{N}, \mathcal{L} \cup \mathcal{S})$, where $\mathcal{L} \cap \mathcal{S} = \emptyset$, $\mathcal{L}$ is the set of *local* edges and $\mathcal{S}$ is the set of *shortcut* edges. The edges in $\mathcal{L}$ satisfy the following property:

Property 1. For every pair of distinct objects $x, t \in \mathcal{N}$ there exists an object u such that $(x, u) \in \mathcal{L}$ and $u \prec_t x$.

I.e., for any x and t, x has a local edge leading to an object closer to t.

A comparison oracle can be used to route a message from s to t over the edges in graph G. In particular, given graph G, we define *greedy forwarding* [9] over G as follows. Let $\Gamma(s)$ be the neighborhood of s, *i.e.*, $\Gamma(s) = \{u \in \mathcal{N} \text{ s.t. } (s, u) \in \mathcal{L} \cup \mathcal{S}\}$. Given a source s and a target t, greedy forwarding sends a message to neighbor w of s that is as close to t as possible, *i.e.*, $w \in \min_{\preccurlyeq_t} \Gamma(s)$. If $w \neq t$, the above process is repeated at w; if $w = t$, greedy forwarding terminates. Property 1 guarantees that greedy forwarding from any source s will eventually reach t: there is always a neighbor closer to t than the object/node forwarding the message. Moreover, if the message is at x, the closest neighbor w can be found using $|\Gamma(x)|$ queries to a comparison oracle.

The edges in $\mathcal{L}$ are called "local" because they are typically determined by object proximity. For example, in the classic paper by Kleinberg [9], objects are arranged uniformly in a rectangular k-dimensional grid—with no gaps—and d is taken to be the Manhattan distance on the grid. Moreover, there exists an $r \geq 1$ such that

$$\mathcal{L} = \{(x, y) \in \mathcal{N} \times \mathcal{N} \text{ s.t. } d(x, y) \leq r\}. \tag{3}$$

Assuming every position in the rectangular grid is occupied, such edges indeed satisfy Property 1. In this work, we *do not* require that edges in $\mathcal{L}$ are given by any locality-based definition like (3); our only assumption is that they satisfy Property 1. Nevertheless, for consistency, we also refer to edges in $\mathcal{L}$ as "local".

Our goal is to select the shortcut edges in $\mathcal{S}$ so that greedy forwarding is as efficient as possible. In particular, assume that we can select no more than β shortcut edges, where β is a positive integer. For S a subset of $\mathcal{N} \times \mathcal{N}$ such that $|S| \leq \beta$, we denote by $C_S(s, t)$ the cost of greedy forwarding, in message hops, for forwarding a message from s to t given that $\mathcal{S} = S$. We allow the selection of shortcut edges to be random: the set $\mathcal{S}$ can be a random variable over all subsets S of $\mathcal{N} \times \mathcal{N}$ such that $|S| \leq \beta$. We denote the distribution of $\mathcal{S}$ by $\Pr(\mathcal{S} = S)$ for $S \subseteq \mathcal{N} \times \mathcal{N}$ such that $|S| \leq \beta$. Given a source s and a target t, let $\mathbb{E}[C_{\mathcal{S}}(s, t)] = \sum_{S \subseteq \mathcal{N} \times \mathcal{N} : |S| \leq \beta} C_S(s, t) \cdot \Pr(\mathcal{S} = S)$ be the expected cost of forwarding a message from s to t with greedy forwarding, in message hops.

We consider again a heterogeneous demand: a source and target object are selected at random from $\mathcal{N} \times \mathcal{N}$ according to a demand probability distribution λ. The *small-world network design problem* can then be formulated as follows.

> Small-World Network Design (SWND): Given an embedding of $\mathcal{N}$ into $(\mathcal{M}, d)$, a set of local edges $\mathcal{L}$, a demand distribution λ, and an integer $\beta > 0$, select a r.v. $\mathcal{S} \subset \mathcal{N} \times \mathcal{N}$, where $|\mathcal{S}| \leq \beta$, that minimizes $\bar{C}_{\mathcal{S}} = \sum_{(s,t) \in \mathcal{N} \times \mathcal{N}} \lambda(s, t) \mathbb{E}[C_{\mathcal{S}}(s, t)]$.

In other words, we wish to select $\mathcal{S}$ so that the cost of greedy forwarding is minimized. Note again that the free variable of SWND is the distribution of $\mathcal{S}$.

5 Main Results

We now present our main results with respect to SWND and CSTC. Our first result is negative: optimizing greedy forwarding is a hard problem.

Theorem 1. *SWND is NP-hard.*

The proof of this theorem can be found in our technical report [7]. In short, the proof reduces DominatingSet to the decision version of SWND. Interestingly, the reduction is to a SWND instance in which (a) the metric space is a 2-dimensional grid, (b) the distance metric is the Manhattan distance on the grid and (c) the local edges are given by (3). Thus, SWND remains NP-hard even in the original setup considered by Kleinberg [9].

The NP-hardness of SWND suggests that this problem cannot be solved in its full generality. Motivated by this, as well as its relationship to content search through comparisons, we focus our attention to the following version of the SWND problem, in which we place additional restrictions to the shortcut edge set $\mathcal{S}$. First, $|\mathcal{S}| = |\mathcal{N}|$, and for every $x \in \mathcal{N}$ there exists *exactly one* shortcut edge $(x, y) \in \mathcal{S}$. Second, the object y to which x connects is selected independently at each x, according to a probability distribution $\ell_x(y)$. *I.e.*, for $\mathcal{N} = \{x_1, x_2, \ldots, x_n\}$, the joint distribution of shortcut edges has the form:

$$\Pr(\mathcal{S} = \{(x_1, y_1), \ldots (x_n, y_n)\}) = \prod_{i=1}^{n} \ell_{x_i}(y_i). \tag{4}$$

We call this version of the SWND problem the *one edge per object* version, and denote it by 1-SWND. Note that, in 1-SWND, the free variables are the distributions ℓ_x, $x \in \mathcal{N}$.

For a given demand λ, recall that μ is the marginal distribution of the demand λ over the target set $\mathcal{T}$, and that for $A \subset \mathcal{N}$, $\mu(A) = \sum_{x \in A} \mu(x)$. Then, for any two objects $x, y \in \mathcal{N}$, we define the *rank* of object y w.r.t. object x as follows:

$$r_x(y) \equiv \mu(B_x(d(x, y))) \tag{5}$$

where $B_x(r)$ is the closed ball with radius r centered at x.

Suppose now that shortcut edges are generated according to the joint distribution (4), where the outgoing link from an object $x \in \mathcal{N}$ is selected according to the following probability:

$$\ell_x(y) \propto \frac{\mu(y)}{r_x(y)}, \tag{6}$$

for $y \in \mathsf{supp}(\mu)$, while for $y \notin \mathsf{supp}(\mu)$ we define $\ell_x(y)$ to be zero. Eq. (6) implies the following appealing properties. For two objects y, z that have the same distance from x, if $\mu(y) > \mu(z)$ then $\ell_x(y) > \ell_x(z)$, *i.e.*, y has a higher probability of being connected to x. When two objects y, z are equally likely to be targets, if $y \prec_x z$ then $\ell_x(y) > \ell_x(z)$. The distribution (6) thus biases

both towards objects close to x as well as towards objects that are likely to be targets. Finally, if the metric space $(\mathcal{M}, d)$ is a k-dimensional grid and the targets are uniformly distributed over $\mathcal{N}$ then $\ell_x(y) \propto (d(x,y))^{-k}$. This is the shortcut distribution used by Kleinberg in [9]; (6) is thus a generalization of this distribution to heterogeneous targets as well as to more general metric spaces.

Our next theorem, whose proof is in Section 5.1, relates the cost of greedy forwarding under (6) to the entropy H, the max-entropy $H_{\max}$ and the doubling parameter c of the target distribution μ.

Theorem 2. *Given a demand λ, consider the set of shortcut edges $\mathcal{S}$ sampled according to (4), where $\ell_x(y)$, $x, y \in \mathcal{N}$, are given by (6). Then*

$$\bar{C}_{\mathcal{S}} \leq 6c^3(\mu) \cdot H(\mu) \cdot H_{\max}(\mu).$$

Note that the bound in Theorem 2 depends on λ only through the target distribution μ. In particular, it holds for *any* source distribution ν, and *does not require* that sources are selected independently of the targets t. Moreover, if $\mathcal{N}$ is a k-dimensional grid and μ is the uniform distribution over $\mathcal{N}$, the above bound becomes $O(\log^2 n)$, retrieving thus Kleinberg's result [9].

Exploiting an underlying relationship between 1-SWND and CSTC, we can obtain an efficient selection policy for greedy content search. In particular,

Theorem 3. *Given a demand λ, consider the memoryless selection policy $\Pr(\mathcal{F}(\mathcal{H}_k, x_k) = w) = \ell_{x_k}(w)$ where ℓ_x is given by (6). Then*

$$\bar{C}_{\mathcal{F}} \leq 6c^3(\mu) \cdot H(\mu) \cdot H_{\max}(\mu).$$

The proof of this theorem is almost identical, *mutatis mutandis*, to the proof of Theorem 2, and can be found in our technical report [7]. Like Theorem 2, Theorem 3 characterises the search cost in terms of the doubling constant, the entropy and the max-entropy of μ. This is very appealing, given (a) the relationship between $c(\mu)$ and the topology of the target set and (b) the classic result regarding the entropy and accesses to a membership oracle, as outlined in Section 3.

The distributions ℓ_x are defined in terms of the embedding of $\mathcal{N}$ in $(\mathcal{M}, d)$ and the target distribution μ. Interestingly, however, the bounds of Theorem 3 can be achieved if *neither* the embedding in $(\mathcal{M}, d)$ *nor* the target distribution μ are a priori known. In our technical report [7] we propose an adaptive algorithm that asymptotically achieves the performance guarantees of Theorem 3 only through access to a comparison oracle. In short, the algorithm learns the ranks $r_x(y)$ and the target distribution μ as searches through comparisons take place.

A question arising from Theorems 2 and 3 is how tight these bounds are. Intuitively, we expect that the optimal shortcut set $\mathcal{S}$ and the optimal selection policy $\mathcal{F}$ depend both on the entropy of the target distribution and on its doubling constant. Our next theorem, whose proof is in Section 5.2, establishes that this is the case for $\mathcal{F}$.

Theorem 4. *For any integer K and D, there exists a metric space $(\mathcal{M}, d)$ and a target measure μ with entropy $H(\mu) = K\log(D)$ and doubling constant $c(\mu) = D$ such that the average search cost of any selection policy $\mathcal{F}$ satisfies*

$$\bar{C}_{\mathcal{F}} \geq H(\mu)\frac{c(\mu)-1}{2\log(c(\mu))}. \tag{7}$$

Hence, the bound in Theorem 3 is tight within a $c^2(\mu)\log(c(\mu))H_{\max}$ factor.

5.1 Proof of Theorem 2

According to (6), the probability that object x links to y is given by $\ell_x(y) = \frac{1}{Z_x}\frac{\mu(y)}{r_x(y)}$, where $Z_x = \sum_{y\in\mathcal{T}}\frac{\mu(y)}{r_x(y)}$ is a normalization factor bounded as follows.

Lemma 1. *For any $x \in \mathcal{N}$, let $x^* \in \min_{\preccurlyeq_x}\mathcal{T}$ be any object in $\mathcal{T}$ among the closest targets to x. Then $Z_x \leq 1 + \ln(1/\mu(x^*)) \leq 3H_{\max}$.*

Proof. Sort the target set $\mathcal{T}$ from the closest to furthest object from x and index objects in an increasing sequence $i = 1,\ldots,k$, so the objects at the same distance from x receive the same index. Let A_i, $i = 1,\ldots,k$, be the set containing objects indexed by i, and let $\mu_i = \mu(A_i)$ and $\mu_0 = \mu(x)$. Furthermore, let $Q_i = \sum_{j=0}^{i}\mu_j$. Then $Z_x = \sum_{i=1}^{k}\frac{\mu_i}{Q_i}$. Define $f_x(r) : \mathbb{R}^+ \to \mathbb{R}$ as $f_x(r) = \frac{1}{r} - \mu(x)$. Clearly, $f_x(\frac{1}{Q_i}) = \sum_{j=1}^{i}\mu_j$, for $i \in \{1,2\ldots,k\}$. This means that we can rewrite Z_x as $Z_x = \sum_{i=1}^{k}(f_x(1/Q_i) - f_x(1/Q_{i-1}))/Q_i$. By reordering the terms involved in the sum above, we get $Z_x = f_x(\frac{1}{Q_k})/Q_k + \sum_{i=1}^{k-1} f_x(1/Q_i)(\frac{1}{Q_i} - \frac{1}{Q_{i+1}})$. First note that $Q_k = 1$, and second that since $f_x(r)$ is a decreasing function, $Z_x \leq 1 - \mu_0 + \int_{1/Q_k}^{1/Q_1} f_x(r)dr = 1 - \frac{\mu_0}{Q_1} + \ln\frac{1}{Q_1}$. This shows that if $\mu_0 = 0$ then $Z_x \leq 1 + \ln\frac{1}{\mu_1}$ or otherwise $Z_x \leq 1 + \ln\frac{1}{\mu_0}$. □

Given the set $\mathcal{S}$, recall that $C_{\mathcal{S}}(s,t)$ is the number of steps required by the greedy forwarding to reach $t \in \mathcal{N}$ from $s \in \mathcal{N}$. We say that a message at object v is in phase j if $2^j\mu(t) \leq r_t(v) \leq 2^{j+1}\mu(t)$. Notice that the number of different phases is at most $\log_2 1/\mu(t)$. We can write $C_{\mathcal{S}}(s,t)$ as

$$C_{\mathcal{S}}(s,t) = X_1 + X_2 + \cdots + X_{\log\frac{1}{\mu(t)}}, \tag{8}$$

where X_j are the hops occurring in phase j.Assume that $j > 1$, and let $I = \left\{w \in \mathcal{N} : r_t(w) \leq \frac{r_t(v)}{2}\right\}$. The probability that v links to an object in the set I, and hence moving to phase $j-1$, is $\sum_{w\in I}\ell_{v,w} = \frac{1}{Z_v}\sum_{w\in I}\frac{\mu(w)}{r_v(w)}$. Let $\mu_t(r) = \mu(B_t(r))$ and $\rho > 0$ be the smallest radius such that $\mu_t(\rho) \geq r_t(v)/2$. Since we assumed that $j > 1$ such a $\rho > 0$ exists. Clearly, for any $r < \rho$ we have $\mu_t(r) < r_t(v)/2$. In particular, $\mu_t(\rho/2) < \frac{1}{2}r_t(v)$. On the other hand, since the doubling parameter is $c(\mu)$ we have $\mu_t(\rho/2) > \frac{1}{c(\mu)}\mu_t(\rho) \geq \frac{1}{2c(\mu)}r_t(v)$. Therefore,

$$\frac{1}{2c(\mu)}r_t(v) < \mu_t(\rho/2) < \frac{1}{2}r_t(v). \tag{9}$$

Let $I_\rho = B_t(\rho)$ be the set of objects within radius $\rho/2$ from t. Then $I_\rho \subset I$, so $\sum_{w\in I} \ell_{v,w} \geq \frac{1}{Z_v}\sum_{w\in I_\rho}\frac{\mu(w)}{r_v(w)}$. By triangle inequality, for any $w \in I_\rho$ and y such that $d(y,v) \leq d(v,w)$ we have $d(t,y) \leq \frac{5}{2}d(v,t)$. This means that $r_v(w) \leq \mu_t(\frac{5}{2}d(v,t))$, and consequently, $r_v(w) \leq c^2(\mu)r_t(v)$. Therefore, $\sum_{w\in I}\ell_{v,w} \geq \frac{1}{Z_v}\frac{\sum_{w\in I_\rho}\mu(w)}{c^2(\mu)r_t(v)} = \frac{1}{Z_v}\frac{\mu_t(\rho/2)}{c^2(\mu)r_t(v)}$. By (9), the probability of terminating phase j is uniformly bounded by

$$\sum_{w\in I}\ell_{v,w} \geq \min_v \frac{1}{2c^3(\mu)Z_v} \overset{\text{Lem. 1}}{\geq} \frac{1}{6c^3(\mu)H_{\max}(\mu)} \tag{10}$$

As a result, the probability of terminating phase j is stochastically dominated by a geometric random variable with the parameter given in (10). This is because (a) if the current object does not have a shortcut edge which lies in the set I, by Property 1, greedy forwarding sends the message to one of the neighbours that is closer to t and (b) shortcut edges are sampled independently across neighbours. Hence, given that t is the target object and s is the source object,

$$\mathbb{E}[X_j|s,t] \leq 6c^3(\mu)H_{\max}(\mu). \tag{11}$$

Suppose now that $j = 1$. By the triangle inequality, $B_v(d(v,t)) \subseteq B_t(2d(v,t))$ and $r_v(t) \leq c(\mu)r_t(v)$. Hence, $\ell_{v,t} \geq \frac{1}{Z_v}\frac{\mu(t)}{c(\mu)r_t(v)} \geq \frac{1}{2c(\mu)Z_v} \geq \frac{1}{6c(\mu)H_{\max}(\mu)}$ since object v is in the first phase and thus $\mu(t) \leq r_t(v) \leq 2\mu(t)$. Consequently,

$$\mathbb{E}[X_1|s,t] \leq 6c(\mu)H_{\max}(\mu). \tag{12}$$

Combining (8), (11), (12) and using the linearity of expectation, we get $\mathbb{E}[C_{\mathcal{S}}(s,t)] \leq 6c^3(\mu)H_{\max}(\mu)\log\frac{1}{\mu(t)}$ and, thus, $\bar{C}_{\mathcal{S}} \leq 6c^3(\mu)H_{\max}(\mu)H(\mu)$. □

5.2 Proof of Theorem 4

Our proof amounts to constructing a metric space and a target distribution μ for which the bound holds. Our construction will be as follows. For some integers D, K, the target set $\mathcal{N}$ is taken as $\mathcal{N} = \{1,\dots,D\}^K$. The distance $d(x,y)$ between two distinct elements x, y of $\mathcal{N}$ is defined as $d(x,y) = 2^m$, where

$$m = \max\{i \in \{1,\dots,K\} : x(K-i) \neq y(K-i)\}.$$

We then have the following

Lemma 2. *Let μ be the uniform distribution over $\mathcal{N}$. Then (i) $c(\mu) = D$, and (ii) if the target distribution is μ, the optimal average search cost C^* based on a comparison oracle satisfies $C^* \geq K\frac{D-1}{2}$.*

Before proving Lemma 2, we note that Thm. 4 immediately follows as a corollary.

Proof (of Lemma 2). Part (i): Let $x = (x(1),\dots x(K)) \in \mathcal{N}$, and fix $r > 0$. Assume first that $r < 2$; then, the ball $B(x,r)$ contains only x, while the ball

$B(x, 2r)$ contains either only x if $r < 1$, or precisely those $y \in \mathcal{N}$ such that $(y(1), \ldots, y(K-1)) = (x(1), \ldots, x(K-1))$ if $r \geq 1$. In the latter case $B(x, 2r)$ contains precisely D elements. Hence, for such $r < 2$, and for the uniform measure on $\mathcal{N}$, the inequality

$$\mu(B(x, 2r)) \leq D\mu(B(x, r)) \tag{13}$$

holds, and with equality if in addition $r \geq 1$.

Consider now the case where $r \geq 2$. Let the integer $m \geq 1$ be such that $r \in [2^m, 2^{m+1})$. By definition of the metric d on $\mathcal{N}$, the ball $B(x, r)$ consists of all $y \in \mathcal{N}$ such that $(y(1), \ldots, y(K-m)) = (x(1), \ldots, x(K-m))$, and hence contains $D^{\min(K,m)}$ points. Similarly, the ball $B(x, 2r)$ contains $D^{\min(K,m+1)}$ points. Hence (13) also holds when $r \geq 2$.

Part (ii): We assume that the comparison oracle, in addition to returning one of the two proposals that is closer to the target, also reveals the distance of the proposal it returns to the target. We further assume that upon selection of the initial search candidate x_0, its distance to the unknown target is also revealed. We now establish that the lower bound on C^* holds when this additional information is available; it holds a fortiori for our more resticted comparison oracle.

We decompose the search procedure into phases, depending on the current distance to the destination. Let L_0 be the integer such that the initial proposal x_0 is at distance 2^{L_0} of the target t, i.e. $(x_0(1), \ldots, x_0(K-L_0)) = (t(1), \ldots, t(K-L_0))$, $x_0(K-L_0+1) \neq t(K-L_0+1)$. No information on t can be obtained by submitting proposals x such that $d(x, x_0) \neq 2^{L_0}$. Thus, to be useful, the next proposal x must share its $(K-L_0)$ first components with x_0, and differ from x_0 in its $(K-L_0+1)$-th entry. Now, keeping track of previous proposals made for which the distance to t remained equal to 2^{L_0}, the best choice for the next proposal consists in picking it again at distance 2^{L_0} from x_0, but choosing for its $(K-L_0+1)$-th entry one that has not been proposed so far. It is easy to see that, with this strategy, the number of additional proposals after x_0 needed to leave this phase is uniformly distributed on $\{1, \ldots D-1\}$, the number of options for the $(K-L_0+1)$-th entry of the target.

A similar argument entails that the number of proposals made in each phase equals 1 plus a uniform random variable on $\{1, \ldots, D-1\}$. It remains to control the number of phases. We argue that it admits a Binomial distribution, with parameters $(K, (D-1)/D)$. Indeed, as we make a proposal which takes us into a new phase, no information is available on the next entries of the target, and for each such entry, the new proposal makes a correct guess with probability $1/D$. This yields the announced Binomial distribution for the numbers of phases (when it equals 0, the initial proposal x_0 coincided with the target).

Thus the optimal number of search steps C verifies $C \geq \sum_{i=1}^{X}(1+Y_i)$, where the Y_i are i.i.d., uniformly distributed on $\{1, \ldots, D-1\}$, and independent of the random variable X, which admits a Binomial distribution with parameters $(K, (D-1)/D)$. Thus using Wald's identity, we obtain that $\mathbb{E}[C] \geq \mathbb{E}[X]\mathbb{E}[Y_1]$, which readily implies (ii). □

Note that the lower bound in (ii) has been established for search strategies that utilize the entire search history. Hence, it is *not* restricted to memoryless search.

6 Conclusions

In this work, we initiated a study of CTSC and SWND under heterogeneous demands, tying performance to the topology and the entropy of the target distribution. Our study leaves several open problems, including improving upper and lower bounds for both CSTC and SWND. Given the relationship between these two, and the NP-hardness of SWND, characterizing the complexity of CSTC is also interesting. Also, rather than considering restricted versions of SWND, as we did here, devising approximation algorithms for the original problem is another possible direction.

Earlier work on comparison oracles eschewed metric spaces altogether, exploiting what where referred to as *disorder inequalities* [5, 11, 12]. Applying these under heterogeneity is also a promising research direction. Finally, trade-offs between space complexity and the cost of the learning phase vs. the costs of answering database queries are investigated in the above works, and the same trade-offs could be studied in the context of heterogeneity.

References

[1] Clarkson, K.L.: Nearest-neighbor searching and metric space dimensions. In: Shakhnarovich, G., Darrell, T., Indyk, P. (eds.) Nearest-Neighbor Methods for Learning and Vision: Theory and Practice, pp. 15–59. MIT Press, G. Shakhnarovich (2006)
[2] Cover, T.M., Thomas, J.: Elements of Information Theory. Wiley, Chichester (1991)
[3] Fraigniaud, P., Giakkoupis, G.: On the searchability of small-world networks with arbitrary underlying structure. In: STOC (2010)
[4] Fraigniaud, P., Lebhar, E., Lotker, Z.: A doubling dimension threshold $\theta(\log \log n)$ for augmented graph navigability. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 376–386. Springer, Heidelberg (2006)
[5] Goyal, N., Lifshits, Y., Schutze, H.: Disorder inequality: a combinatorial approach to nearest neighbor search. In: WSDM (2008)
[6] Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: STOC, pp. 604–613 (1998)
[7] Karbasi, A., Ioannidis, S., Massoulie, L.: Content search through comparisons. Tech. Rep. CR-PRL-2010-07-0002, Technicolor (2010)
[8] Karger, D., Ruhl, M.: Finding nearest neighbors in growth-restricted metrics. In: SODA (2002)
[9] Kleinberg, J.: The small-world phenomenon: An algorithmic perspective. In: STOC (2000)
[10] Krauthgamer, R., Lee, J.R.: Navigating nets: simple algorithms for proximity search. In: SODA (2004)
[11] Lifshits, Y., Zhang, S.: Combinatorial algorithms for nearest neighbors, near-duplicates and small-world design. In: SODA (2009)
[12] Tschopp, D., Diggavi, S.N.: Approximate nearest neighbor search through comparisons (2009)
[13] Tschopp, D., Diggavi, S.N.: Facebrowsing: Search and navigation through comparisons. In: ITA Workshop (2010)
[14] White, R., Roth, R.: Exploratory Search: Beyond the Query-Response Paradigm. Morgan & Claypool (2009)

Efficient Distributed Communication in Ad-Hoc Radio Networks*

Bogdan S. Chlebus[1], Dariusz R. Kowalski[2], Andrzej Pelc[3], and Mariusz A. Rokicki[2]

[1] Department of Computer Science and Engineering, U. of Colorado Denver, USA
[2] Department of Computer Science, University of Liverpool, UK
[3] Département d'informatique, Université du Québec en Outaouais, Canada

Abstract. We present new distributed deterministic solutions to two communication problems in n-node ad-hoc radio networks: rumor gathering and multi-broadcast. In these problems, some or all nodes of the network initially contain input data called rumors, which have to be learned by other nodes. In rumor gathering, there are k rumors initially distributed arbitrarily among the nodes, and the goal is to collect all the rumors at one node. Our rumor gathering algorithm works in $\mathcal{O}((k+n)\log n)$ time and our multi-broadcast algorithm works in $\mathcal{O}(k\log^3 n + n\log^4 n)$ time, for any n-node networks and k rumors (with arbitrary k), which is a substantial improvement over the best previously known deterministic solutions to these problems.

As a consequence, we *exponentially* decrease the gap between upper and lower bounds on the deterministic time complexity of four communication problems: rumor gathering, multi-broadcast, gossiping and routing, in the important case when every node has initially at most one rumor (this is the scenario for gossiping and for the usual formulation of routing). Indeed, for $k = \mathcal{O}(n)$, our results simultaneously decrease the complexity gaps for these four problems from polynomial to polylogarithmic in the size of the graph. Moreover, our *deterministic* gathering algorithm applied for $k = \mathcal{O}(n)$ rumors, improves over the best previously known *randomized* algorithm of time $\mathcal{O}(k\log n + n\log^2 n)$.

Keywords: radio network, distributed algorithm, gossiping, gathering, multi-broadcast, routing.

1 Introduction

Radio networks are defined by the property that one wave frequency is used for wireless communication. Multiple messages arriving at a node in the same round

* The first author is supported by the NSF Grant 1016847. The second author is supported by the Engineering and Physical Sciences Research Council [grant numbers EP/G023018/1, EP/H018816/1]. The third author is partially supported by a NSERC discovery grant and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 613–624, 2011.

interfere with one another so that none can be heard. We use simple undirected graphs to represent network topologies. Each of our algorithms works for any network topology on n nodes. The performance bounds we develop hold for all networks of a given size. Nodes are not aware of the topology of the network.

We present distributed deterministic algorithms for two communication tasks: rumor gathering and multi-broadcast. In these problems, some or all nodes of the network initially contain input data called *rumors*, which have to be learned by other nodes. Messages are *small*: each of them carries at most one rumor and $\mathcal{O}(\log n)$ additional control bits. For rumor gathering, there are k rumors, where k is an arbitrary positive integer (independent on n), initially distributed arbitrarily among the nodes, and the goal is to collect all the rumors at one node. Multi-broadcast is related to two fundamental communication problems: gossiping and routing. In gossiping, every node is initialized with a rumor and the goal is for all nodes to learn all rumors. In routing, k rumors distributed arbitrarily among the nodes must be delivered each to its designated destination. The problem of multi-broadcast, considered in this work, is defined as follows: some k rumors are distributed arbitrarily among the nodes and the goal is for all the nodes to learn every rumor. This is a generalization of gossiping (initially not all nodes need to have a rumor) and a strengthening of routing (all nodes need to learn each rumor, not only the destination node for a given rumor).

The challenge of the considered problems is due to the combination of two features of the model of ad-hoc radio networks. On the one hand, nodes do not know the topology of the network, and on the other hand communication algorithms are hard to design due to possible collisions that prevent nodes from hearing messages. In unknown topologies collisions are unavoidable, and nodes need not even be aware of their occurrence, as they do not have the collision detection capability.

Our approach to overcome these problems builds on two main algorithmic techniques introduced in this paper, both of them being deterministic and fully distributed. First, we build a tree to gather and spread rumors (c.f., Section 3.1). This tree, called a Breadth-Then-Depth tree, has the property that it can be clustered into a logarithmic number of forests, and in each forest messages can be propagated in parallel without causing any collision except possibly in roots of the forest. Moreover, each path of this tree from a node to the root is split into an at most logarithmic number of sub-paths, each belonging to a different forest (c.f., Section 4). Note that a DFS tree could cause a large number of collisions while propagating rumors within a path to the root, due to possible shortcut edges, which disallows any parallelism in the general case. A BFS tree, in turn, could cause a problem with a potentially large number of collisions when transmitting from one layer to another, and it is not even known how to construct it deterministically and in a distributed way in a sub-quadratic number of rounds in general radio networks. The second crucial technique is to efficiently propagate rumors between different forests of the constructed clustering, which we do by computing (again, in a deterministic and distributed way) short transmission

schedules between the roots and leaves of neighboring forests (c.f., Section 5). We also resolve a number of technical obstacles, such as how to propagate rumors on paths and how to deal with potential collisions in roots of the forests.

Our Results. Building on novel algorithmic techniques introduced in this work we develop efficient deterministic distributed algorithms for gathering rumors and for multi-broadcast.

Our rumor gathering algorithm works in $\mathcal{O}((k+n)\log n)$ time and our multi-broadcast algorithm works in $\mathcal{O}(k\log^3 n + n\log^4 n)$ time, for any n-node networks and k rumors (with arbitrary k), which is a substantial improvement over the best previously known deterministic solutions to these problems. Indeed, the best previously known deterministic algorithms for the problems of gathering, multi-broadcast and routing worked in time $\mathcal{O}(\min\{(k+n)n^{1/2}\log^3 n, (k+n^{5/2})\log^3 n\})$ [6, 9]. In the case of the gossiping problem, using our multi-broadcast algorithm (for $k=n$) we accomplish gossiping in $\mathcal{O}(n\log^4 n)$ time, while the best algorithm for this problem worked in time $\mathcal{O}(n^{3/2}\log^3 n)$ [6]. On the other hand, the best known deterministic lower bound for the first three problems was $\Omega(k+n\log n)$ [5, 8, 10, 12], while the best lower bound for gossiping was $\Omega(n\log n)$ [10].

As a consequence, we decrease *exponentially* the gap between upper and lower bounds on the deterministic time complexity of four communication problems: rumor gathering, multi-broadcast, gossiping and routing, in the important case when every node has initially at most one rumor (this is the scenario for gossiping and for the usual formulation of routing). Indeed, for $k=\mathcal{O}(n)$, our results simultaneously decrease the complexity gaps for these four problems from polynomial to polylogarithmic in the size of the graph.

Moreover, our *deterministic* gathering algorithm applied for $k=\mathcal{O}(n)$ rumors, improves over the best previously known *randomized* algorithm [3] of time $\mathcal{O}(k\log n + n\log^2 n)$.

Relying on our multi-broadcast algorithm, every node can learn the whole network topology in time $\mathcal{O}(m\log^3 n + n\log^4 n)$, where m is the number of edges in the network; learning topology is a possible preparation to switching from ad-hoc algorithms to centralized ones. The best previous algorithm for this task, based on gossip from [6], guaranteed only $\mathcal{O}((m+n)n^{1/2}\log^3 n)$ time.

Related Work. Algorithmic aspects of radio communication have been widely studied in the last two decades. Both the directed graph model [4, 7, 11] and the undirected (symmetric) graph model [2, 12] of this paper have been used. Broadcasting [1, 2, 4, 8] and various variants of many-to-many communication primitives [3, 5–7, 10] were considered for radio networks.

Below we give the best previously known results for four communication problems in ad hoc radio networks with which we are concerned in this paper: gathering, gossiping, routing and multi-broadcast.

The best previously known deterministic algorithm for the problems of gathering, routing and multi-broadcast worked in time $\mathcal{O}(\min\{(k+n)n^{1/2}\log^3 n, (k+n^{5/2})\log^3 n\})$ [6, 9], where the first part of the formula follows from a direct

application of the gossip algorithm from [6] to consecutive batches of rumors, while the second part of the formula comes from learning the network topology (i.e., $\mathcal{O}(n^2)$ edges) by gossiping from [6] and then applying multi-broadcast techniques for known topology graphs as in [9]. The best randomized algorithm used expected time $\mathcal{O}((k + n \log n) \log n)$ [3]. The best deterministic lower bound for these problems was $\Omega(k + n \log n)$ [5, 8, 10, 12].

For gossiping, the best previously known deterministic algorithm worked in time $\mathcal{O}(n^{3/2} \log^3 n)$ [6] and the best randomized algorithm used expected time $\mathcal{O}(n \log^2 n)$ [3]. The best deterministic lower bound was $\Omega(n \log n)$ [10].

2 Preliminaries

We model a radio network as a simple undirected connected graph $G = (V, E)$, in which nodes represent stations and an edge connects two nodes that can directly transmit to each other. There are n nodes. Each node has a unique integer label assigned to it from the range $[1, N]$, for some positive integer N. We assume that $N = \mathcal{O}(n^\gamma)$ for a constant $\gamma > 1$, so that $\log N = \mathcal{O}(\log n)$, and that both n and N are powers of 2. (The latter assumption is for convenience only and can be easily removed.) All logarithms are to the base 2. We may assume that the nodes know only an upper bound on the number of nodes that is $\mathcal{O}(n)$. Here 'knowledge' means that the respective information can be part of code of an algorithm. We consider *distributed algorithms for ad hoc networks*, i.e., algorithms are not supplied with any a priori information about the topology of the underlying network. Also the number k of rumors is not initially known.

Communication proceeds in synchronous rounds. A node can transmit or receive one message per round. A message transmitted by a node reaches all its neighbors in the round of the transmission. A message is said to be *heard* when it has been received successfully. Radio networks are defined by the property that a node hears a message in a round precisely when the node acts as a receiver and exactly one of its neighbors transmits in this round. If at least two neighbors of a node u transmit simultaneously in a given round, then none of the messages is heard by u and we say that a *collision* occurred at u. We do not assume that a node can distinguish between collision and 'silence' which occurs when no neighbor transmits.

The inputs initially available to nodes are called *rumors*. We assume that all rumors are of equal size, and also that all messages are of equal size (otherwise we could pad a default sequence of bits to make them equal). In this paper we work in the model of *small* messages, in which it takes one whole message to carry a rumor. A message is capable of carrying up to $\mathcal{O}(\log n)$ additional control bits.

All procedures and algorithms, as well as their analysis, are given assuming an upper bound polynomial in n on the number k of rumors known to the nodes. This assumption is only for the sake of simplicity, and can be easily removed without increasing asymptotic complexity. The missing details and proofs are deferred to the full version of the paper.

For an integer $m > 0$, the notation $[m]$ denotes the set $\{1, \ldots, m\}$. A *transmission schedule* $\mathcal{F} = (F_0, \ldots, F_{\ell-1})$ is a sequence of subsets of $[N]$. We say that

$\mathcal{F}$ is *executed* in a sequence of consecutive rounds when the nodes transmitting in round i are precisely those whose labels are in F_k, where k and i are congruent modulo ℓ.

A sequence $\mathcal{F}(m,k) = (F_0, \ldots, F_{\ell-1})$ of subsets $F_i \subseteq [m]$ is a (m,k)-*selective family* of length ℓ if, for each subset $S \subseteq [m]$ of at most k elements, there exists $F_j \in \mathcal{F}(m,k)$ such that $|F_j \cap S| = 1$. Selective families are used to produce efficient transmission schedules. It was proved in [8] that for any positive integers $m \geq k$ there exists an (m,k)-selective family of length $\mathcal{O}(k \log(m/k))$. Note that the round-robin transmission sequence, where all sets F_j are singletons, is a selective family but it is not efficient, since N is polynomial in n.

3 Building Blocks

We begin by developing building blocks for our target algorithms.

3.1 Breadth-Then-Depth Search (BTD)

We design a new token traversal of the network. The token is propagated through neighboring nodes, starting from the root. The goal of the token is to visit all the nodes and return to the root. We call the traversal *breadth-then-depth (BTD)* and the resulting tree is called a BTD tree. The concept of a BTD traversal is similar to that of DFS. The difference is that in an instance of passing the token from the token's holder to its unvisited neighbor only one edge is added to the DFS tree (i.e., the edge connecting both these nodes), while in the construction of BTD all neighbors of the token's holder that are outside of the current BTD tree get connected by an edge to the token's holder, and thus to the tree.

In order to implement the BTD traversal in a radio network, we rely on a construction to assign to any node v one of its neighbors; such a neighbor of v will be referred to as the *offshoot of* v. Given a node v and its designated neighbor w, it is possible to find an offshoot of v distinct from w, if there is at least one such neighbor, by a collaboration between v and w that terminates in $\mathcal{O}(\log N)$ time. It was shown in [12] how to accomplish locating an offshoot. The node w is called the *witness of* v in such a context.

BTD search from a designated root can be accomplished by the following procedure BTD_Search. Nodes have status either *visited* or *unvisited*. Once a node becomes visited, its status never changes. In the beginning all nodes except the root are unvisited, while the root is visited and holds the token.

Let $\mathcal{F}$ be a (N,n)-selective family of length $cn \log N \subseteq \mathcal{O}(n \log n)$, for some constant $c > 0$, which exists by [8]. BTD_Search proceeds as follows.

In the first $cn \log N$ rounds, the family $\mathcal{F}$ is executed by all nodes except the root, with each node transmitting its own label. After that, the root transmits the first label w it heard, in order to notify this neighbor w that it has just become a witness of the root. Next, the following procedure is repeated until termination. Suppose that node v holds a token and w is its (neighbor) witness; we assume that v is visited. If v is not the root then v also has a designated parent.

- Node v holding the token transmits a control message in order to let all its unvisited neighbors that have not yet heard a control message select v as their parent. From the moment when a node receives a control message for the first time and selects its parent we call this node *joined*. All nodes that selected v as their parent did it in the same time — just after v had received the token for the first time and sent the control message.
- Node v holding the token cooperates with its witness w to find an offshoot among the unvisited children of v in the constructed BTD tree. This process takes a fixed time $\mathcal{O}(\log N)$. If there is no unvisited child of v, in which case an offshoot is not selected, and v is the root then the procedure terminates.
- If an offshoot v^* has been selected then node v passes the token to v^*; otherwise v passes the token to its parent in the constructed BTD tree.
- If a joined node v^*, selected as an offshoot, receives a token, then v^* becomes visited, and v becomes its witness. The node v is also a parent of v^* in the constructed BTD tree.

Analysis. The following property follows from the design of the procedure, from the logarithmic time bound on locating an offshoot [12], and from the $\mathcal{O}(n \log n)$ length of the selective family [8] used for choosing a witness of the root.

Lemma 1. *Procedure* BTD_Search *performs a BTD search on the whole network and spans a BTD tree rooted at the initiating node in* $\mathcal{O}(n \log n)$ *rounds.*

It can be shown that a BTD tree can be clustered into paths so that no collisions are caused by processes propagating messages on "parallel" paths, and there is an at most logarithmic number of paths on any branch from the root to a leaf.

3.2 Computing Weights on a Tree

Given a rooted tree D, with rumors located at nodes of this tree, we define the *weight of node* v as the sum of two numbers: the number of nodes in the subtree rooted at v and the number of rumors located in nodes of this subtree. The weight of the root of tree D is called the *weight of tree* D. The following procedure Compute_Weights(D) computes the weight of each node of D locally at this node. The input tree D is given locally: each node knows only its parent and children (if any) in the tree. The procedure is initiated by the root of tree D: a token is issued by the root and travels along the tree in a DFS manner, eventually coming back to the root. The token carries two counters. The first counter is initialized to 0 and is incremented by $r + 1$ at each first visit of a node carrying r rumors. Each node computes its weight by subtracting the value of the counter of the first visit of the token from the value of the last visit. The second counter is initialized to 0 and incremented by one each time a node is visited by the token for the first time in this run. This counter is used for enumerating all nodes from 0 to $n - 1$.

Lemma 2. *Algorithm* Compute_Weights(D) *terminates in time* $2k$*, where* k *is the number of nodes of* D*.*

3.3 Rumor Spreading on a Path

For a simple path P, let its length $|P|$ be equal the number of edges, so that there are $|P|+1$ nodes on the path. Let x be an upper bound on the weight of P interpreted as a tree, which means that there are at most $x-|P|-1$ rumors in the nodes of P. We assume that each node of path P knows x, that P is given locally in the sense that each node in path P knows its neighbors in P, and that one end point of path P is designated to be a root. It is also assumed that there are no links connecting any two nodes in path P that are not neighbors in P. We develop a procedure called Spread_on_Path(P,x) whose execution results in every node in path P receiving all rumors originally located in the nodes on P.

The procedure consists of three parts. In the first part the root initiates a token that traverses all the nodes in path P and carries a tally of the already visited nodes. This allows each node in P to learn its distance from the root of P. The first part takes up to x rounds, but if less than x then the nodes wait idle through round x so that all nodes in P know when part one ends. Immediately afterwards the *up-propagation part* starts to take the next $4x$ rounds; again the nodes wait idle through round $4x$ if needed. The goal of this part is to deliver all rumors in P to the root of P. In round i of this part, each node with the distance from the root congruent to $3-i$ modulo 3 transmits the first packet in its transmission queue. In this queue there are packets that have not been transmitted by this node in the previous rounds of this procedure; the queue operates as a stack (LIFO queue). If there is no pending packet, then a node pauses and does not transmit any messages. Nodes that are not scheduled to transmit in a round automatically listen in this round to hear transmissions of neighbors. If a node receives a packet from a neighbor that is further away from the root then the recipient node enqueues the new packet to be eventually transmitted in due order. When the second part is concluded then the root has gathered all rumors. In the third part the other end of the path becomes a *new root*. The third part follows: it proceeds similarly as the second part with the modification that only the rumors stored in the original root are propagated towards the new root. This third part is called *down-propagation* also takes precisely $4x$ rounds, with nodes idling through round $4x$ if needed.

Lemma 3. *Consider an execution of procedure* Spread_on_Path(P,x). *If the sum of the number of nodes in path P and the number of rumors is at most x, then all rumors initially located in nodes of path P are delivered to all nodes of P within $9x$ rounds.*

4 Gathering Rumors

In the problem of gathering, there are k different rumors distributed arbitrarily among n nodes of the network and the goal is to gather all the rumors in one designated node called the *sink*. The gathering problem is handled by introducing and exploring a clustering based on newly defined BTD search trees and a weighing method. The key underlying property of this approach is that we can

quickly and locally identify sufficiently long and disjoint paths, and then pipeline the rumors along these paths without causing interference. We also show that only a logarithmic number of changes between paths is sufficient to transmit a rumor from its source to the root of the tree.

We start with describing the clustering procedure Cluster_to_Gather. Next we give the main procedure Tree_Gather to gather rumors in the root of a given BTD tree known locally, exploring the computed clustering. We conclude with the final description of algorithm Radio_Gather.

In the specification of the procedures Cluster_to_Gather and Tree_Gather we assume that there is an exact number of rounds dedicated to each part, and even though the activity in some part may be finished earlier, nodes wait until the end of the specified period before starting the next part. This assumption allows to synchronize parts of these procedures and simplify the analysis.

Clustering Procedure. We introduce a procedure Cluster_to_Gather(D, x), which has input parameters with the following properties:

G1: A rooted BTD tree D spanning V with each node knowing only its parent and children, if any, in the tree; in particular, a node knows whether it is the root or not;

G2: An upper bound x on the sum of the number of rumors and of $|V|$; x is known to every node; there is an initial allocation of at most $x - |V| \geq 1$ rumors among the nodes in V.

The procedure computes a clustering of a given BTD tree D in network G. This clustering is a generalization of heavy path decomposition (c.f., [13]) to weighted trees in the radio network model. Together with some properties of BTD tree, the clustering allows an efficient parallelization of gathering and spreading rumors in radio networks. Assume that the properties G1-G2 are satisfied by D and by the initial rumor allocation. To simplify the exposition, we can assume without loss of generality that x is a power of 2. Define S_i, for an integer $0 \leq i \leq \log x$, to be the set of nodes in V weighing more than 2^{i-1} and at most 2^i in tree D. (Nodes in S_0 do not have any rumor in the beginning.) Let D_i be the sub-forest of the considered component induced by the nodes in S_i; we will show later that it is a collection of paths of tree D. For each connected component of D_i we define its root as the (only) node that does not have its parent in this component, and its *second-in-charge* node as the (only) child of the root in the component (if the component has more than one node).

Lemma 4. *Consider an execution of the procedure Cluster_to_Gather(D, x). Upon its termination, which happens after $6x$ rounds, each node knows to which set S_i it belongs, and whether it is a root or a second-in-charge node. Moreover, for any $0 \leq i \leq \log x$, the set S_i and the corresponding forest D_i satisfy the following properties:*

(i) D_i consists of paths, each being a sub-path of some path from the root to some leaf in the initial tree D; moreover, the sub-graph of G induced by the nodes on such a single path forms the same path (i.e., there are no shortcuts between nodes in the path in the underlying network G);

(ii) for every two path-components of forest D_i, if two nodes from different components are connected by an edge in graph G, at least one of them must be the root of one component.

Procedure. Cluster_to_Gather(D, x)

Part 1: The root in D calls procedure Compute_Weights(D). Upon completion of this part each node in V locally computes its weight, which is in the range $[x]$, and its number in the traversal route, which is in the range $[n]$. This part takes $2x$ rounds.

Part 2: The root in D initiates a token to traverse the tree D in a DFS manner. The token propagates the weight of the current owner of the token computed during Part 1. Upon completion of this part each node knows in which sets S_i are its children and its parent in tree D. This part takes $2x$ rounds.

Part 3: The root in D initiates a token to traverse the tree D in a DFS manner. It carries the information whether the parent of the current owner of the token is in the same set S_i — this was computed by the sender during Part 2. Upon completion of this part each node knows whether it is second-in-charge in some tree-component of the corresponding forest D_i or not. This part takes $2x$ rounds.

Gathering Procedure. Given a rooted tree D, we design the gathering procedure along this tree as follows:

Procedure. Tree_Gather(D, x)

Part 1: Procedure Cluster_to_Gather(D, x) is executed.

Part 2: for $i = 1$ to $\log x$ do: perform the stages

Stage (a): Each node v second-in-charge in S_i initiates Spread_on_Path(P, x) as the root of path P which is defined as follows. A node is in P, if it is in S_i and is a descendant of v in the tree D. This stage takes $9x$ rounds.

Stage (b): The root of D initiates the token, which is sent along tree D in a DFS manner. A node whose parent in D is in a set S_j for $j \neq i$ is a *root of the forest* D_i. Whenever a root of the forest D_i receives a traversing token for the first time, it keeps forwarding all its gathered rumors towards its grandparent in D, one after another. Each such forwarding operation takes two rounds and goes through the parent of the token holder. After the last rumor is delivered, the token is passed to the next node in the DFS traversal of D. This stage takes $4x$ rounds.

Part 3: The root of D initiates procedure Spread_on_Path$(S_{\log x}, x)$. This part takes $9x$ rounds.

Lemma 5. *An execution of procedure* Tree_Gather(D, x) *results in gathering all the rumors stored in the nodes of the tree D in the root of D, which takes $\mathcal{O}(x \log n)$ rounds, if x is an upper bound on the sum of the number of nodes $|V|$ and the number of rumors initially stored in S.*

Algorithm. Finally we specify an algorithm for the gathering problem, with a designated sink node w:

Algorithm. Radio_Gather(w)

Part 1: Node w calls procedure BTD_Search to create a BTD tree D spanning the network G and rooted at w.
Part 2: Node w calls procedure Compute_Weights(D), and then computes its weight x.
Part 3: Procedure Tree_Gather(D, x) is executed.

Theorem 1. *Algorithm Radio_Gather solves the gathering problem for any distribution of k rumors in a n-node network, using $\mathcal{O}((k+n)\log n)$ rounds.*

5 Multi-Broadcast

Multi-broadcast is a generalization of gossiping: there are k rumors distributed arbitrarily among nodes and the goal is for all the nodes to learn every rumor.

Graph Clustering. Procedure Cluster_to_Spread(S, D) is similar to procedure Cluster_to_Gather(S, D) from Section 4. The differences are as follows: (i) while pre-computing a partitioning of the nodes and a BTD tree D into sets S_i and forests D_i, for $0 \leq i \leq \log n$, the weight is defined only in terms of the number of nodes, that is, we do not account for any rumors in the nodes; (ii) apart from computing by each node its weight and the weight of all its neighbors in the network, and thus being able to recognize to which set S_i the node and its neighbors belong, each node v computes a specific subset R_v of its neighbors.

Dense Broadcast Schedules. Define a (n, a, ε)-*ultra-selector* to be a family of sets such that for any set $A \subseteq [n]$ of size at most a and more than $a/2$ there is at least a ε fraction of the sets in the family that intersect A on a single element, for a given $1 \leq a \leq n$ and $0 < \varepsilon \leq 1$.

Lemma 6. *For any given constant $0 < \varepsilon \leq 1/32$ and sufficiently large parameter n, there is a constant $\beta > 1$, which depends only on ε, such that for every $1 \leq a \leq n$ there exists a (n, a, ε)-ultra-selector of length at most $\beta \cdot a \log(2n/a)$.*

Procedure Partial_Ultra_Broadcast is composed of $\log n$ interleaved threads, where in the i-th thread the $(n, 2^i, 1/32)$-ultra-selector is executed in a cyclic way, for $0 \leq i \leq \log n - 1$; in thread $\log n$ the round robin procedure on range $[n]$ is run (observe that the round robin transmission scheme corresponds to a $(n, n, 1/32)$-ultra-selector). The length of each thread is bounded by $\beta \cdot n$, where β and the ultra-selectors are as specified in Lemma 6. Therefore, the whole length of procedure Partial_Ultra_Broadcast is $\beta \cdot n \log n$. It follows from Lemma 6, when applied to the neighborhood of a node, that for any node there is at least a fraction $\frac{1}{32 \log n}$ of the rounds of Partial_Ultra_Broadcast that are successful from the point of view of this node, in the sense that in each of these rounds there is exactly one neighbor of this node scheduled to transmit.

Computing Short Broadcast Schedules for Pipelining Rumors. This is the most involved technical part of our multi-broadcast algorithm. Suppose that we are given a set $S^* \subseteq V$ and the spanning tree D of network G. We assume that each node locally knows D and can locally check whether itself and some of its neighbors in G are in set S^*. The aim is to construct a family $\{\mathcal{B}_v\}_{v \in S^*}$ that is a broadcast schedule from set S^* to set S' of children of S^* in the tree D. By a broadcast schedule we understand a set of transmission schedules such that each node in S' eventually receives a message from some node in S^*, if nodes in S^* follow the schedule. The following Procedure Down_Hop(S^*, D), for any subset $S^* \subseteq S$, outputs in each node v a subset of round-numbers $\mathcal{B}_v \subseteq \{1, 2, \ldots, 64 \log^2 n\}$.

The procedure is recursive with respect to the set S' of uninformed children of nodes in S^* in tree D. Additionally, nodes maintain the depth of recursion, called *depth*. In the very beginning of the procedure, each node $w \in S'$ of some node in set S^* simulates locally procedure Partial_Ultra_Broadcast with respect to all its neighbors in S^* in network G; based on this, node w computes the set of rounds $\mathcal{R}_w$ in which it would successfully receive a message from some of its neighbors in S^*. The first $2n \log(\beta n \log^2 n)$ communication rounds are split into $\log(\beta n \log^2 n)$ consecutive parts, each consisting of $2n$ communication rounds in which a token traverses the whole tree D in a DFS manner, computing and updating local values. The goal of these parts is to find a round $b \in \{1, \ldots, \beta n \log^2 n\}$ in the locally simulated execution of procedure Partial_Ultra_Broadcast in which at least $\frac{|S'|}{32 \log n}$ nodes in S' would receive a message from some of their neighbors in S^*.

Consequently, all nodes in S^* that were scheduled to transmit in round b during procedure Partial_Ultra_Broadcast are set as transmitters in round *depth* of the schedules $\{\mathcal{B}_v\}_{v \in S^*}$ output by procedure Down_Hop(S^*, D), i.e., $depth \in \mathcal{B}_v$ for any $v \in S^*$ that was scheduled to transmit in round b of the (locally simulated) execution of procedure Partial_Ultra_Broadcast. By the properties of procedure Partial_Ultra_Broadcast, set S' shrinks by the fraction $\frac{1}{32 \log n}$. After that, we increase *depth* by one and call recursively Down_Hop(S^*, D) with respect to the new set S' of uninformed children of S^* in D. It follows that after at most $64 \log^2 n$ recursive calls, set S' becomes empty, and therefore the output schedules $\{\mathcal{B}_v\}_{v \in S^*}$ are well defined. The length of the procedure is $\mathcal{O}(\log^2 n \cdot n \log n) \subseteq \mathcal{O}(n \log^3 n)$.

Pipelining Rumors from a Source. We specify procedure Tree_Spread(D, k) which, given a BTD spanning tree D and k rumors gathered in the root of the tree, delivers the rumors to all nodes in the network. The procedure is initiated by the root. First, all nodes in path $D_{\log n}$ execute procedure Spread_on_Path$(S_{\log n}, n + k)$, so that all of them get all k rumors from the root. Then $\log n$ stages are executed sequentially. The goal of stage i, for $i = \log n, \log n - 1, \ldots, 1$, is to deliver rumors gathered by each node of forest D_i to every node in forest D_{i-1}. This is achieved by applying twice procedure Down_Hop and procedure Spread_on_Path. The whole propagation of rumors from nodes in D_i to all nodes in D_{i-1} takes $\mathcal{O}(n \log^3 n + k \log^2 n)$ rounds. There are $\log n$ forests, due to clustering, and therefore the whole cost of procedure Tree_Spread(D, k) is $\mathcal{O}(n \log^4 n + k \log^3 n)$.

Multi-Broadcast Algorithm. We use the leader election algorithm described in [7] (Radio_Leader), to find a leader in time $\mathcal{O}(n \log^3 n)$ using $\mathcal{O}(\log n)$-bit messages.

Algorithm. Radio_Multi_Broadcast

Part 1: Run algorithm Radio_Leader to elect a leader w.
Part 2: Gather all the rumors in w by calling algorithm Radio_Gather(w). Create a BTD tree D, spanning network G and rooted at w, and distribute the number k of rumors gathered in w.
Part 3: Node w calls procedure Tree_Spread(V, D, k).

Theorem 2. *Algorithm Radio_Multi_Broadcast completes multi-broadcast for any distribution of k rumors in a n-node radio network in time $\mathcal{O}(k \log^3 n + n \log^4 n)$.*

Corollary 1. *Routing k rumors in a n-node network can be accomplished in time $\mathcal{O}(k \log^3 n + n \log^4 n)$, and gossiping can be accomplished in time $\mathcal{O}(n \log^4 n)$.*

References

1. Alon, N., Bar-Noy, A., Linial, N., Peleg, D.: A lower bound for radio broadcast. Journal of Computer and System Sciences 43, 290–298 (1991)
2. Bar-Yehuda, R., Goldreich, O., Itai, A.: On the time complexity of broadcast in radio networks: An exponential gap between determinism and randomization. Journal of Computer and System Sciences 45, 104–126 (1992)
3. Bar-Yehuda, R., Israeli, A., Itai, A.: Multiple communication in multi-hop radio networks. SIAM Journal on Computing 22, 875–887 (1993)
4. Chlebus, B.S., Gąsieniec, L., Gibbons, A.M., Pelc, A., Rytter, W.: Deterministic broadcasting in ad hoc radio networks. Distributed Computing 15, 27–38 (2002)
5. Chlebus, B.S., Kowalski, D.R., Radzik, T.: Many-to-many communication in radio networks. Algorithmica 54(1), 118–139 (2009)
6. Christersson, M., Gąsieniec, L., Lingas, A.: Gossiping with bounded size messages in ad-hoc radio networks. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 377–389. Springer, Heidelberg (2002)
7. Chrobak, M., Gąsieniec, L., Rytter, W.: Fast broadcasting and gossiping in radio networks. Journal of Algorithms 43, 177–189 (2002)
8. Clementi, A.E.F., Monti, A., Silvestri, R.: Distributed broadcasting in radio networks of unknown topology. Theoretical Computer Science 302, 337–364 (2003)
9. Gąsieniec, L., An, H.-C., Pelc, A., Xin, Q.: Deterministic M2M multicast in radio networks. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 670–682. Springer, Heidelberg (2004)
10. Gąsieniec, L., Potapov, I.: Gossiping with unit messages in known radio networks. In: Proc. of the 2nd International Conference on Theoretical Computer Science (TCS), pp. 193–205 (2002)
11. Gąsieniec, L., Radzik, T., Xin, Q.: Faster deterministic gossiping in directed ad-hoc radio networks, in. In: Proc. of the 9th Scandinavian Workshop on Algorithm Theory (SWAT), pp. 397–407 (2004)
12. Kowalski, D.R., Pelc, A.: Broadcasting in undirected ad-hoc radio networks. Distributed Computing 18, 43–57 (2005)
13. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. Journal of Computer and System Sciences 26, 362–391 (1983)

Nearly Optimal Bounds for Distributed Wireless Scheduling in the SINR Model

Magnús M. Halldórsson and Pradipta Mitra

ICE-TCS, School of Computer Science
Reykjavik University
101 Reykjavik, Iceland
mmh@ru.is, ppmitra@gmail.com

Abstract. We study the wireless scheduling problem in the physically realistic SINR model. More specifically: we are given a set of n links, each a sender-receiver pair. We would like to schedule the links using the minimum number of slots, using the SINR model of interference among simultaneously transmitting links. In the basic problem, all senders transmit with the same uniform power.

In this work, we provide a distributed $O(\log n)$-approximation for the scheduling problem, matching the best ratio known for centralized algorithms. This is based on an algorithm studied by Kesselheim and Vöcking, improving their analysis by a logarithmic factor. We show this to be best possible for any such distributed algorithm.

Our analysis extends also to linear power assignments, and as well as for more general assignments, modulo assumptions about message acknowledgement mechanisms.

Keywords: Distributed Scheduling, Wireless Networks, SINR Model.

1 Introduction

Given a set of n wireless links, each a sender-receiver pair, what is the minimum number of slots needed to schedule all the links, given interference constraints? This is the canonical *scheduling* problem in wireless communication that we study here in a distributed setting.

In a wireless network, communication occurring simultaneously in the same channel interfere with each other. Algorithmic questions for wireless networks depend crucially on the model of interference considered. In this work, we use the physical model or "SINR model" of interference, precisely defined in Section 2. It is known to capture reality more faithfully than the graph-based models most common in the theory literature, as shown theoretically as well as experimentally [8,15,17]. Early work on scheduling in the SINR model focused on heuristics and/or non-algorithmic average-case analysis (e.g. [9]). In seminal work, Moscibroda and Wattenhofer [16] proposed the study of the *scheduling complexity* of arbitrary set of wireless links. Numerous works on various problems in the SINR setting have appeared since ([5,7,12,3,11], to point out just a few).

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 625–636, 2011.

The *scheduling* problem has primarily been studied in a centralized setting. In many realistic scenarios, however, it is imperative that a distributed solution be found, since a centralized controller may not exist, and individual nodes in the link may not be aware of the overall topology of the network. For the scheduling problem, the only known rigorous result is due to Kesselheim and Vöcking [14], who show that a simple and natural distributed algorithm provides $O(\log^2 n)$ approximation to the scheduling problem.

In this work, we adopt the same algorithm as Kesselheim and Vöcking, but provide an improved analysis of $O(\log n)$-approximation. Moreover, we show that this is essentially the best result obtainable by a distributed algorithm.

2 Preliminaries and Contributions

Given is a set $L = \{\ell_1, \ell_2, \ldots, \ell_n\}$ of links, where each link ℓ_v represents a communication request from a sender s_v to a receiver r_v. The distance between two points x and y is denoted $d(x, y)$. The asymmetric distance from link ℓ_v to link ℓ_w is the distance from v's sender to w's receiver, denoted $d_{vw} = d(s_v, r_w)$. The length $d(s_v, r_v)$ of link ℓ_v is simply denoted by ℓ_v.

Let P_v denote the power assigned to link ℓ_v, or, in other words, s_v transmits with power P_v. We adopt the *physical model* (or *SINR model*) of interference, in which a receiver r_v successfully receives a message from a sender s_v if and only if the following condition holds:

$$\frac{P_v/\ell_v^\alpha}{\sum_{\ell_w \in S\setminus\{\ell_v\}} P_w/d_{wv}^\alpha + N} \geq \beta, \tag{1}$$

where N is a universal constant denoting the ambient noise, $\alpha > 0$ denotes the path loss exponent, $\beta \geq 1$ denotes the minimum SINR (signal-to-interference-noise-ratio) required for a message to be successfully received, and S is the set of concurrently scheduled links in the same *slot*. We say that S is *SINR-feasible* (or simply *feasible*) if (1) is satisfied for each link in S.

Given a set of links L, the **scheduling problem** is to find a partition of L of minimum size such that each subset in the partition is feasible. The size of the partition equals the minimum number of slots required to schedule all links. We will call this number the **scheduling number** of L, and denote it by $T(L)$ (or T when clear from context).

The above defines the physical model for uni-directional links. In the *bi-directional* setting [5], the asymmetry between senders and receivers disappear. The SINR constraint (1) changes only in that the definition of distance between links changes to $d_{wv} = \min(d(s_w, r_v), d(s_v, r_w), d(s_w, s_v), d(r_w, r_v))$. With this new definition of inter-link distances, all other definitions and conditions remain unchanged. We will focus on the uni-directional model, but our proofs easily extend to the bi-directional case.

A power assignment P is **length-monotone** if $P_v \geq P_w$ whenever $\ell_v \geq \ell_w$ and **sub-linear** if $\frac{P_v}{\ell_v^\alpha} \leq \frac{P_w}{\ell_w^\alpha}$ whenever $\ell_v \geq \ell_w$. Two widely used power

assignments in this class are the *uniform* power assignment, where every link transmits with the same power; and the *linear* power assignment, where P_v is proportional to ℓ_v^α.

Distributed algorithms. In the traditional distributed setting, a communication infrastructure exists which can be used to run distributed algorithms. The current setting is different, where the goal is to construct such an infrastructure. Thus our algorithm will work with very little global knowledge and minimal external input. We assume that the senders and receivers have a rough estimate of the network size n, have synchronized clocks, and are given fixed length-monotone, sub-linear power assignments that they must use.

The algorithm works by having the senders repeatedly transmitting until they succeed, necessitating an acknowledgement from receivers to the senders, so that the sender would know when to stop. We assume that these acknowledgements are the only external information that the senders receive. Any algorithm that works under these constraints will be referred to as an **ack-only** algorithm. We study the scheduling problem under two separate assumptions about acknowledgements. In the first model, we require the algorithm to generate explicit acknowledgements. In the second model, we assume that acknowledgements are "free" or much cheaper than data packets, thus do not have to be explicitly realized by the algorithm.

Affectance. We will use the notion of *affectance*, introduced in [7,12] and refined in [14] to the thresholded form used here, which has a number of technical advantages. The affectance $a_w^P(v)$ *on* link ℓ_v *from* another link ℓ_w, with a given power assignment P, is the interference of ℓ_w on ℓ_v relative to the power received, or

$$a_w^P(v) = \min\left\{1, c_v \frac{P_w/d_{wv}^\alpha}{P_v/\ell_v^\alpha}\right\} = \min\left\{1, c_v \frac{P_w}{P_v} \cdot \left(\frac{\ell_v}{d_{wv}}\right)^\alpha\right\},$$

where $c_v = \beta/(1 - \beta N \ell_v^\alpha/P_v)$ is a constant depending only on the length and power of the link ℓ_v. We will drop P when it is clear from context. Let $a_v(v) = 0$. For a set S of links and a link ℓ_v, let $a_v(S) = \sum_{\ell_w \in S} a_v(w)$ and $a_S(v) = \sum_{\ell_w \in S} a_w(v)$. For sets S and R, $a_R(S) = \sum_{\ell_v \in R} \sum_{\ell_u \in S} a_v(u)$. Using such notation, Eqn. 1 can be rewritten as $a_S(v) \leq 1$, and **this is the form we will use**.

Signal-strength and robustness. A δ-*signal* set is one where the affectance on any link is at most $1/\delta$. A set is SINR-feasible iff it is a 1-signal set. We know:

Lemma 1 ([10]). *Let ℓ_u, ℓ_v be links in a q^α-signal set. Then, $d_{uv} \cdot d_{vu} \geq q^2 \cdot \ell_u \ell_v$.*

2.1 Our Contributions and Related Work

We achieve the following results:

Theorem 1. *There is a $O(\log n)$-approximate ack-only distributed algorithm for the scheduling problem for uniform and linear power assignments, as well as for*

all length-monotone sub-linear power assignments for bi-directional links. If we additionally assume free (or cheap) acknowledgements the same result holds for all length-monotone sub-linear power assignments for uni-directional links.

Theorem 2. *Assuming that all senders use an identical algorithm, no distributed ack-only algorithm can approximate the scheduling problem by a factor better than $\Omega(\log n)$, even with free acknowledgements.*

As in [14], our results hold in arbitrary distance metrics (and do not require the common assumption that $\alpha > 2$).

The scheduling problem has been profitably studied in the *centralized* setting by a number of works. The problem is known to be NP-hard [7]. For length-monotone, sub-linear power assignments, a $O(\log n)$ approximation for general metrics has been achieved recently [11] following up on earlier work [7,12]. In the birectional setting with power control, Fanghänel et al. [5] provided a $O(\log^{3.5+\alpha} n)$ approximation algorithm, recently improved to $O(\log n)$ [10,11]. For linear power on the plane, [6] provides an additive approximation algorithm of $O(T + \log^2 n)$. On the plane, $O(\log n)$ approximation for power control for uni-directional links has been recently achieved [13]. Chafekar et al. [3] provide a $O(\log^2 \Delta \log^2 \Gamma \log n)$ approximation to the joint multi-hop scheduling and routing problem.

In the distributed setting, the related *capacity* problem (where one wants to find the maximum subset of L that can be transmitted in a single slot) has been studied a series of papers [1,4,2], and have culminated in a $O(1)$-approximation algorithm for uniform power [2]. However, these game-theoretic algorithms take time polynomial in n to converge, thus can be seen more appropriately to determine capacity, instead of realizing it in "real time".

For distributed scheduling, the only work we are aware of remains the interesting paper by Kesselheim and Vöcking [14], who give a $O(\log^2 n)$ distributed approximation algorithm for the scheduling problem with any fixed length-monotone and sub-linear power assignment. They consider the model with no free acknowledgements, however their results do not improve if free acknowledgements are assumed. Thus in all cases considered, our results constitute a $\Omega(\log n)$ factor improvement.

In [14], the authors introduce a versatile measure, the maximum average affectance $\bar{A}$, defined by

$$\bar{A} = \max_{R \subseteq L} \text{avg}_{\ell_u \in R} \sum_{\ell_v \in R} a_v(u) = \max_{R \subseteq L} \frac{1}{|R|} \sum_{\ell_u \in R} \sum_{\ell_v \in R} a_v(u) \ .$$

The authors then show two results. On the one hand, they show that $\bar{A} = O(T \log n)$ where $T = T(L)$. On the other hand, they present a natural algorithm (we use the same algorithm in this work) which schedules all links in $O(\bar{A} \log n)$ slots, thus achieving a $O(\log^2 n)$ approximation. We can show (omitted in this version of the paper) that both of these bounds are tight. Thus it is not possible to obtain improved approximation using the measure $\bar{A}$.

Our main technical insight is to devise a different measure that avoids bounding the global average, instead looks at the average over a large fraction of the linkset. This allows us to shave off a $\log n$ factor. The measure $\Lambda(L)$ (or Λ when clear from the context) can be defined as follows:

$$\Lambda(L) = \arg\min_t \text{s.t.} \forall R \subseteq L, |\{\ell \in R : a_R(\ell) \leq 4t\}| \geq |R|/4 \ .$$

To get an intuitive feel for this measure, consider any given R. Since we only insist that a large fraction of R have affectance bounded by t, the value of t may be significantly smaller than the average affectance in R (and as a consequence $\Lambda(L)$ may be much smaller than $\bar{A}$). Indeed, we show that $\Lambda = O(T)$ and that the algorithm schedules all links in time $O(\Lambda \log n)$, achieving the claimed approximation factor. We can give instances where $T(L) = \theta(\Lambda(L) \log n)$, and thus the performance of the algorithm is best possible in that respect.

3 $O(\log n)$-Approximate Distributed Scheduling Algorithm

The algorithm from [14] is listed below as **Distributed**. It is a very natural algorithm, in the same tradition of backoff schemes as ALOHA [18], or more recent algorithms in the SINR model [4,2].

Algorithm 1. Distributed

1: $k \leftarrow 0$
2: **while** transmission not successful **do**
3: $\quad q = \frac{1}{4 \cdot 2^k}$
4: $\quad$ **for** $\frac{8c_3 \ln n}{q}$ slots **do**
5: $\quad\quad$ transmit with i.i.d. probability q
6: $\quad$ **end for**
7: $\quad k \leftarrow k + 1$
8: **end while**

The algorithm is mostly self-descriptive. One point to note is that Line 2 necessitates some sort of acknowledgement mechanism for the distributed algorithm to stop. For simplicity, we will defer the issue of acknowledgements to Section 3.2 and simply assume their existence for now. Theorem 3 below implies our main positive result.

Theorem 3. *If all links of set L run **Distributed**, then each link will be scheduled by time $O(\Lambda(L) \log n)$ with high probability.*

To prove Theorem 3, we claim the following.

Lemma 2. *Given is a request set R with measure $\Lambda = \Lambda(R)$. Consider a time slot in which each sender of the requests in R transmits with probability $q \leq \frac{1}{8\Lambda}$. Then at least $\frac{q \cdot |R|}{8}$ transmissions are successful in expectation.*

Proof. Let $M = \{\ell_u \in R : a_R(u) \leq 4\Lambda\}$. By definition of Λ, $M \geq |R|/4$. It suffices to show that at least $q|M|/2$ transmissions are successful in expectation.

For $\ell_u \in R$, let T_u be indicator random variable that link ℓ_u transmits, and S_u the indicator r.v. that ℓ_u succeeds. Now, $\mathbb{E}(S_u) = \mathbb{P}(S_u = 1) = \mathbb{P}(T_u = 1)\mathbb{P}(S_u = 1|T_u = 1) = q\mathbb{P}(S_u = 1|T_u = 1) = q(1 - \mathbb{P}(S_u = 0|T_u = 1))$. For $\ell_u \in M$,

$$\mathbb{P}(S_u = 0|T_u = 1) \leq \mathbb{P}\left(\sum_{\ell_v \in R} a_v(u)T_v \geq 1\right) \leq \mathbb{E}\left(\sum_{\ell_v \in R} a_v(u)T_v\right)$$
$$= \sum_{\ell_v \in R} a_v(u)\mathbb{E}(T_v) = q \sum_{\ell_v \in R} a_v(u) \leq q \cdot 4\Lambda \leq 1/2 \ .$$

for $q \leq \frac{1}{8\Lambda}$. Therefore, $\mathbb{E}(S_u) \geq \frac{q}{2}$.

The expected number of successful links is thus

$$\mathbb{E}\left(\sum_{\ell_u \in R} S_u\right) = \sum_{\ell_u \in R} \mathbb{E}(S_u) \geq \sum_{\ell_u \in M} \mathbb{E}(S_u) \geq |M| \cdot q/2 \geq |R| \cdot q/8 \ ,$$

as desired. □

Proof (of Thm. 3). Given Lemma 2, the proof of the Theorem essentially follows the arguments in Thms. 2 and 3 of [14].

First consider the running time of the algorithm when in Line 3, q is set to the "right" value $\frac{1}{8\Lambda}$ stipulated in Lemma 2. (Note that since the algorithm increases the probabilities by a factor of 2, it will select the right probability within that factor). Let n_t be the random variable indicating the number of requests that have not been successfully scheduled in the first t time slots.

Lemma 2 implies that $\mathbb{E}(n_{t+1}|n_t = k) \leq k - \frac{q}{8}k$ and thus

$$\mathbb{E}(n_{t+1}) \leq \sum_{k=0}^{\infty} \mathbb{P}(n_t = k) \cdot (1 - q/8)k = (1 - q/8)\mathbb{E}(n_t) \ .$$

Setting $n_0 = n$, this yields $\mathbb{E}(n_t) \leq (1 - q/8)^t n$. Now, after $8c_3 \log n/q$ time slots for a large enough c_3, the expected number of remaining requests is

$$\mathbb{E}(n_{8c_3 \log n/q}) \leq (1 - q/8)^{8c_3 \log n/q} n \leq \left(\frac{1}{e}\right)^{c_3 \log n} n = n^{1-c_3} \ .$$

By Markov inequality $\mathbb{P}(n_{8c_3 \log n/q} \neq 0) = \mathbb{P}(n_{8c_3 \log n/q} \geq 1) \leq \mathbb{E}(n_{8c_3 \log n/q}) \leq n^{1-c_3}$. So we need $O(\log n/q) = O(\Lambda \log n)$ time, with high probability.

Now we need to show that running the algorithm with the "wrong" values of q doesn't cost too much. This holds because the costs of all previous executions form a geometric series increasing the overall time required by no more than a constant factor (see Thm. 3 of [14] for the argument, which is fairly simple). □

3.1 Bounding the Measure

We will assume that the implicit power assignment is length-monotone and sublinear. We need two related Lemmas to get a handle on affectances. The first of these Lemmas is known.

Lemma 3 (Lemma 7, [14]). *If L is a feasible set, and ℓ_u is a link such that $\ell_u \leq \ell_v$ for all $\ell_v \in L$, then $a_L(u) = O(1)$.*

Now we prove the following.

Lemma 4. *If L is a feasible set such that $a_u(L) \leq 2$ for all $\ell_u \in L$, and ℓ_v is a link such that $\ell_v \leq \ell_u$ for all $\ell_u \in L$, then $a_v(L) = O(1)$.*

Before we prove this below, note that this Lemma can be contrasted with Lemma 9 of [14], which only gives a $O(\log n)$ bound (without the extra condition $a_u(L) \leq 2$). Intuitively, the power of this Lemma is that while $a_u(L)$ can be as large as $O(\log n)$ for feasible L, for a large subset of L the extra condition $a_u(L) \leq 2$ holds (as will be easily shown in Lemma 5), for which the stronger $O(1)$ upper bound will apply.

Proof. [of Lemma 4] We use the signal strengthening technique of [12]. For this, we decompose the set L to $\lceil 2 \cdot 3^\alpha/\beta \rceil^2$ sets, each a 3^α-signal set. We prove the claim for one such set; since there are only constantly many such sets, the overall claim holds. Let us reuse the notation L to be such a 3^α-signal set.

Consider the link $\ell_u = (s_u, r_u) \in L$ such that $d(s_v, s_u)$ is minimum. Also consider the link $\ell_w = (s_w, r_w) \in L$ such that $d(r_w, s_v)$ is minimum. Let $D = d(s_v, s_u)$. We claim that for all links, $\ell_x = (s_x, r_x) \in L, \ell_x \neq \ell_w$,

$$d(s_v, r_x) \geq \frac{1}{2} D \ . \tag{2}$$

To prove this, assume, for contradiction, that $d(s_v, r_x) < \frac{1}{2}D$. Then, $d(s_v, r_w) < \frac{1}{2}D$, by definition of ℓ_w. Now, again by the definition of ℓ_u, $d(s_x, s_v) \geq D$ and $d(s_w, s_v) \geq D$. Thus $\ell_w \geq d(s_v, s_w) - d(s_v, r_w) > \frac{D}{2}$ and similarly $\ell_x > \frac{D}{2}$. On the other hand $d(r_w, r_x) < \frac{D}{2} + \frac{D}{2} < D$. Now, $d_{wx} \cdot d_{xw} \leq (\ell_w + d(r_w, r_x))(\ell_x + d(r_w, r_x)) < (\ell_w + D)(\ell_x + D) < 9\ell_w\ell_x$, contradicting Lemma 1.

By the triangle inquality and Eqn. 2, $d_{ux} = d(s_u, r_x) \leq d(s_u, s_v) + d(s_v, r_x) \leq 3d(s_v, r_x) = 3d_{vx}$. Now $a_v(x) \leq c_x \frac{P_v}{d_{vx}^\alpha} \frac{\ell_x^\alpha}{P_x}$. Since $\ell_v \leq \ell_u$, by length-monotonicity $P_v \leq P_u$. Thus,

$$a_v(x) \leq c_x \frac{P_u}{d_{vx}^\alpha} \frac{\ell_x^\alpha}{P_x} \leq c_x \frac{3^\alpha P_u}{d_{ux}^\alpha} \frac{\ell_x^\alpha}{P_x} = 3^\alpha a_u(x)$$

where the last equality holds because $a_u(x) = c_x \frac{P_u}{d_{ux}^\alpha} \frac{\ell_x^\alpha}{P_x}$ as L is feasible. Finally, summing over all links in L

$$\begin{aligned} a_v(L) &= \sum_{\ell_x \in L} a_v(x) = a_v(w) + \sum_{\ell_x \in L \setminus \{\ell_w\}} a_v(x) \\ &\leq 1 + \sum_{\ell_x \in L \setminus \{\ell_w\}} a_v(x) \leq 1 + 3^\alpha \sum_{\ell_x \in L \setminus \{\ell_w\}} a_u(x) \leq 1 + 3^\alpha \cdot 2 = O(1) \ , \end{aligned}$$

since $\sum_{\ell_x \in L \setminus \{\ell_w\}} a_u(x) \leq a_u(L) \leq 2$ by assumption. □

We can now derive the needed bound on the measure.

Lemma 5. *For any linkset R, $\Lambda(R) \leq c_2 T(R)$ for a fixed constant c_2.*

Proof. It suffices to prove that for every $\hat{R} \subseteq R$, $|\{\ell \in \hat{R} : a_{\hat{R}}(\ell) \leq 4c_2 T\}| \geq \hat{R}/4$. To prove this, the only property we will use of $\hat{R}$ is that $T(\hat{R}) \leq T(R)$, which is obviously true for all $\hat{R}$. Thus, for simplicity we simply prove it for R and the proof will easily carry over to all $\hat{R} \subseteq R$.

Consider a partition of R into T feasible subsets $S_1 \ldots S_T$. For each i, define $S'_i = \{\ell_v \in S_i : a_v(S_i) \leq 2\}$.

Claim. For all i, $|S'_i| \geq \frac{|S_i|}{2}$.

Proof. Since S_i is feasible, the incoming affectance $a_{S_i}(v) \leq 1$ for every link $\ell_v \in S_i$. Let $\hat{S}_i = S_i \setminus S'_i$. Now, $\sum_{v \in \hat{S}_i} a_v(S_i) \leq \sum_{v \in S_i} a_v(S_i) = \sum_{v \in S_i} a_{S_i}(v) \leq |S_i|$. But, $\sum_{v \in \hat{S}_i} a_v(S_i) \geq 2 \cdot |\hat{S}_i|$ by definition of $\hat{S}_i$. Thus, $\hat{S}_i \leq |S_i|/2$, proving the claim. □

Let $R' = \cup_i S'_i$. By the above claim, $|R'| \geq |R|/2$. Let $M = \{\ell_u \in R' : \sum_{\ell_v \in R} a_v(u) \leq 4c_2 T\}$ for some constant c_2. We shall show that $|M| \geq |R'|/2 \geq |R|/4$.

We will prove the claim

$$a_R(R') = c_2 |R| \cdot |T| \ . \tag{3}$$

From this, we get that the average of $a_R(\ell_u)$ over the links $\ell_u \in R'$ is $\frac{c_2|R|\cdot|T|}{|R'|} \leq 2c_2|T|$. At least half of the links in R' have at most double affectance of this average, hence the claim $|M| \geq |R|/4$ and the lemma follow.

Thus our goal becomes to prove Eqn. 3. To prove this note that,

$$a_R(R') = \sum_{i=1}^{T} \sum_{\ell_u \in S'_i} \sum_{j=1}^{T} \sum_{\ell_v \in S_j} a_v(u) = \sum_{i=1}^{T} \sum_{j=1}^{T} a_{S_j}(S'_i) \ . \tag{4}$$

To tackle this sum, we first prove that for any $i, j \leq T$,

$$a_{S_j}(S'_i) \leq O(|S_j| + |S_i|) \ . \tag{5}$$

This holds because,

$$\begin{aligned} a_{S_j}(S'_i) &= \sum_{\ell_u \in S'_i} \sum_{\ell_v \in S_j, \ell_v \geq \ell_u} a_v(u) + \sum_{\ell_u \in S'_i} \sum_{\ell_v \in S_j, \ell_v \leq \ell_u} a_v(u) \\ &\overset{1}{\leq} \sum_{\ell_u \in S'_i} O(1) + \sum_{\ell_u \in S'_i} \sum_{\ell_v \in S_j, \ell_v \leq \ell_u} a_v(u) \leq O(|S'_i|) + \sum_{\ell \in S'_i} \sum_{\ell_v \in S_j, \ell_v \leq \ell_u} a_v(u) \\ &\overset{2}{\leq} O(|S_i|) + \sum_{\ell_v \in S_j} \sum_{\ell_u \in S'_j, \ell_u \geq \ell_v} a_v(u) \overset{3}{\leq} O(|S_i|) + \sum_{\ell_v \in S_j} O(1) \leq O(|S_i|) + O(|S_j|) \ . \end{aligned}$$

Explanation for numbered inequalities:

1. The bound with $O(1)$ follows from Lemma 3.
2. Noting $|S'_i| \leq |S_i|$ for the first term and reorganization of the second term.
3. The $O(1)$ bound is from Lemma 4.

Now we can continue with Eqn. 4 to obtain that

$$a_R(R') = \sum_{i=1}^{T}\sum_{j=1}^{T} a_{S_j}(S'_i) \overset{1}{\leq} \sum_{i=1}^{T}\sum_{j=1}^{T} O(|S_i|+|S_j|) \overset{2}{=} \sum_{i=1}^{T} O(|T|\cdot|S_i|+|R|) = O(|T|\cdot|R|)\ .$$

Explanations for numbered (in)equalities:

1. Due to Eqn. 5.
2. As $\sum_{j=1}^{T} |S_j| = |R|$.

This completes the proof by setting c_2 to be the implicit constant hidden in the Big-O notation. □

3.2 Duality and Acknowledgements

In the above exposition, we ignored the issue of sending acknowledgements from receivers to senders after the communication has succeeded. It is useful to be able to guarantee that acknowledgements would arrive in time $O(T \log n)$, with high probability. Note that for bi-directional links, there is no dichotomy between sender and receiver, so we get our result for all length-monotone, sub-linear power assignments automatically. What follows thus concerns the case of uni-directional links.

The link set L^* associated with the required acknowledgement packets is the "dual" of the original link set L, with the sender and receiver of each link swapping places. Let the dual link associated with a link $\ell_u \in L$ be ℓ^*_u. Now, accommodating the acknowledgements in the same algorithm is not difficult. This can be done, for example, by using the odd numbered slots for the original transmissions, and the even numbered slots for the acknowledgement. It is not obvious, however, that L^* admits a short schedule using a oblivious power assignment.

For uniform power, it was observed in [14] that:

Claim (Observation 12, [14]). Assume both the original and dual link sets use uniform power. Also assume $\ell_u, \ell_v \in L$ can transmit simultaneously. Then, $a_{u^*}(v^*) = \Theta(a_u(v))$, where $a_{u^*}(v^*)$ is the affectance from ℓ^*_u on ℓ^*_v.

Thus the feasibility of S_i implies the near-feasibility of S^*_i. With this observation in hand, the bound follows from arguments of the previous section in a straightforward manner.

For other power assignments, it is easy to show that one cannot in general use the same assignment for the dual. In [14] an elegant approach to this problem was proposed. The authors introduced the notion of **dual power**, where dual link ℓ^*_u uses power $P^*_u = \gamma \frac{\ell_u^\alpha}{P_u}$ (the global normalization factor γ is chosen so that c^*_u is no worse than c_u, for all ℓ_u). In [14] the following is shown:

Claim (Observation 4, [14]). $a_u^P(v) = \Theta(a_{v^*}^{P^*}(u^*))$, for any ℓ_u, ℓ_v in L .

In other words, the incoming affectance of a link ℓ_u in L is close to the outgoing affectance of ℓ_u^* in the "dual" setting (and vice-versa). Using this claim, it is easy to see that the measure $\bar{A}$ (maximum average affectance) is invariant (modulo constants) when switching the problem from L to L^*. Since the authors claim a bound in terms of $\bar{A}$, essentially the same bound can be claimed for both L and L^*.

The situation for our argument is more delicate, since we seek a bound in terms of T, not $\bar{A}$. Unfortunately, T^* (the scheduling number of L^*) cannot be bounded by anything better than $O(T \log n)$, thus a naive application of Thm. 3 only results in a $O(\log^2 n)$ approximation.

However, we can achieve the $O(\log n)$ approximation for linear power, whose dual is uniform power. We define "anti-feasibility" to be: A link set R is *anti-feasible* if $a_u(R) \leq c_4; \forall \ell_u \in R$ for some appropriate constant c_4. Consider now a partition of L into feasible sets $S_1, S_2 \ldots S_T$. By the claim above, it is clear that the sets S_i^* are anti-feasible. Define, $S_i'^* = \{\ell_v^* \in S_i^* : a_{S_i^*}(v^*) \leq 4c_4\}$, $R^* = \cup_{i=1}^T S_i^*$ and $R'^* = \cup_{i=1}^T S_i'^*$. As in the previous section, it is clear that $|R'^*| \geq |R^*|/2$.

We claim that a Lemma similar to Lemma 5 holds, after which $O(T \log n)$ algorithm for the dual set follows the same argument as the rest of the proof of Thm. 3.

Lemma 6. *The following holds for linear power (thus, uniform power for the dual). Let $M^* = \{\ell_v \in R'^* : a_{R^*}(v) \leq 2c_2T\}$, for some constant c_2. Then, $|M^*| \geq |R'^*|/8 \geq |R^*|/16$.*

The crucial thing to notice is that the bound is in terms of T, not the possibly larger T^*.

Proof (Sketch). For uniform power, the following strong result can be proven.

Lemma 7. *If L is an anti-feasible set using uniform power such that $a_L(u) \leq 4c_4$ for ℓ_u, and ℓ_v some other link, also using uniform power, then $a_v(L) = O(1)$.*

This can be proven using techniques we have seen, and has essentially been proven in [2] (Lemma 11). Note that this lemma has no condition on ℓ_v (in terms of being smaller/larger than links of L, unlike Lemmas 3 and 4). Thus this single Lemma suffices in bounding $a_{S_j^*}(S_i'^*)$ *à la Eqn. 5*, from which the whole argument follows (details omitted). □

Finally, if we assume free or cheap acknowledgments, then by Thm. 3, our $O(\log n)$ bound holds for all sub-linear, length-monotone assignments for both uni and bi-directional links. The assumption can be valid in many realistic scenarios, or indeed can guide design decisions. For example, if the size of the acknowledgement packets are smaller than the data packets by a factor of $\Omega(\log n)$, the larger value of T^* is subsumed by having smaller slots for acknowledgement

packets. For illustration, a realistic example of data packet sizes and acknowledgement packet sizes is 1500 bytes and 50 bytes, respectively, and in such a case our bound would hold for networks with up to 2^{30} nodes. Note here that the analysis of [14] is no better than $O(\log^2 n)$ even assuming free acknowledgements.

4 $\Omega(\log n)$-Factor Lower Bound for Distributed Scheduling

We give a construction of $2n$ links on the line that can be scheduled in 2 slots while no distributed algorithm can schedule them in less than $\Omega(\log n)$ slots.

We assume that the distributed algorithm is an **ack-only** algorithm, and that each sender starts at the same time in the same state and uses an identical (randomized) algorithm. Note that the algorithm presented works within these assumptions.

Our construction uses links of equal length, thus the only possible power assignment is uniform. Let $\alpha > 1$, $\beta = 2$ and noise $N = 0$. For the construction, we start with a **gadget** g with two identical links of length 1. Place n such gadgets $g_i, i = 1 \ldots n$ on the line as follows. The two senders of g_i are placed at point $2ni$ and the two receivers of g_i are placed at $2ni + 1$. Now since $\beta = 2$, it is clear that if the two links in the gadget transmit together, neither succeed. On the other hand, links from other gadgets have negligible affectance on a link. To see this, consider the affectance on a link $\ell_u \in g_i$ from all links of other gadgets, i.e., from all links $\ell_v \in \hat{G} = \cup_{j \neq i} g_j$. There are $2n - 2$ links in $\hat{G}$. The distance $d_{vu} \geq \min\{2n(i+1) - 2ni - 1, 2ni + 1 - 2n(i-1)\} = 2n - 1$. Therefore, $\sum_{\hat{G}} a_v(u) \leq (2n-2)\frac{1}{(2n-1)^\alpha} < 1$, for large enough n and $\alpha > 1$. Thus, behavior of links in other gadgets is immaterial to the success of a link. This also implies that the scheduling number of all these links is 2.

To prove the lower bound, let us consider a gadget g_i to be "active" at time t if neither link of g_i succeeded by time $t-1$. Let $T_u(t)$ denote the event that link ℓ_u transmits at time t, and let $A_i(t)$ denote the event that gadget g_i is active at time t.

Lemma 8. *Let ℓ_u and ℓ_v be the identical links in gadget g_i. Then $\mathbb{P}(T_u(t)|A_i(t)) = \mathbb{P}(T_v(t)|A_i(t))$. Moreover, these two probabilities are independent. In other words, the transmission probabilities of two links in a gadget at time t are identical and independent, conditioned on the gadget being active at time t.*

Proof is omitted.

Let this i.i.d. probability be p. Now $\mathbb{P}(A_i(t+1)|A_i(t)) = p^2 + (1-p)^2$, which is minimized for $p = \frac{1}{2}$ with value $\frac{1}{2}$. Thus, $\mathbb{P}(A_i(t+1)|A_i(t)) \geq \frac{1}{2}$.

Theorem 4. $\mathbb{E}(I(n)) = \Omega(\log n)$ *where $I(n)$ is the smallest time at which none of the gadgets are active.*

The full proof is omitted, but intuitively, the bound $\mathbb{P}(A_i(t+1)|A_i(t)) \geq \frac{1}{2}$ implies that on average, no more than half the active gadgets become inactive in a single round, thus it would take $\Omega(\log n)$ rounds for all gadgets to become inactive.

Note that bounding $\mathbb{E}(I(n))$ suffices to lower bound the expected time before all links successfully transmit, since a link cannot succeed as long as the corresponding gadget is active, by definition.

References

1. Andrews, M., Dinitz, M.: Maximizing capacity in arbitrary wireless networks in the SINR model: Complexity and game theory. In: INFOCOM, pp. 1332–1340. IEEE, Los Alamitos (2009)
2. Ásgeirsson, E.I., Mitra, P.: On a game theoretic approach to capacity maximization in wireless networks. In: INFOCOM (2011)
3. Chafekar, D., Kumar, V.S., Marathe, M., Parthasarathy, S., Srinivasan, A.: Cross-layer Latency Minimization for Wireless Networks using SINR Constraints. In: Mobihoc (2007)
4. Dinitz, M.: Distributed algorithms for approximating wireless network capacity. In: INFOCOM, pp. 1397–1405. IEEE, Los Alamitos (2010)
5. Fanghänel, A., Kesselheim, T., Räcke, H., Vöcking, B.: Oblivious interference scheduling. In: PODC, pp. 220–229 (August 2009)
6. Fanghänel, A., Keßelheim, T., Vöcking, B.: Improved algorithms for latency minimization in wireless networks. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 447–458. Springer, Heidelberg (2009)
7. Goussevskaia, O., Halldórsson, M.M., Wattenhofer, R., Welzl, E.: Capacity of Arbitrary Wireless Networks. In: INFOCOM, pp. 1872–1880 (April 2009)
8. Grönkvist, J., Hansson, A.: Comparison between graph-based and interference-based STDMA scheduling. In: Mobihoc, pp. 255–258 (2001)
9. Gupta, P., Kumar, P.R.: The Capacity of Wireless Networks. IEEE Trans. Information Theory 46(2), 388–404 (2000)
10. Halldórsson, M.M.: Wireless scheduling with power control (September 2010); Earlier version appears in ESA 2009, http://arxiv.org/abs/1010.3427
11. Halldórsson, M.M., Mitra, P.: Wireless Capacity with Oblivious Power in General Metrics. In: SODA (2011)
12. Halldórsson, M.M., Wattenhofer, R.: Wireless Communication Is in APX. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 525–536. Springer, Heidelberg (2009)
13. Kesselheim, T.: A Constant-Factor Approximation for Wireless Capacity Maximization with Power Control in the SINR Model. In: SODA (2011)
14. Kesselheim, T., Vöcking, B.: Distributed contention resolution in wireless networks. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 163–178. Springer, Heidelberg (2010)
15. Maheshwari, R., Jain, S., Das, S.R.: A measurement study of interference modeling and scheduling in low-power wireless networks. In: SenSys, pp. 141–154 (2008)
16. Moscibroda, T., Wattenhofer, R.: The Complexity of Connectivity in Wireless Networks. In: INFOCOM (2006)
17. Moscibroda, T., Wattenhofer, R., Weber, Y.: Protocol Design Beyond Graph-Based Models. In: Hotnets (November 2006)
18. Tanenbaum, A.: Computer Networks. Prentice Hall Professional Technical Reference, 4th edn. (2002)

Convergence Time of Power-Control Dynamics*

Johannes Dams, Martin Hoefer, and Thomas Kesselheim

Department of Computer Science,
RWTH Aachen University, Germany
{dams,mhoefer,kesselheim}@cs.rwth-aachen.de

Abstract. We study two (classes of) distributed algorithms for power control in a general model of wireless networks. There are n wireless communication requests or *links* that experience interference and noise. To be successful a link must satisfy an SINR constraint. The goal is to find a set of powers such that all links are successful simultaneously. A classic algorithm for this problem is the fixed-point iteration due to Foschini and Miljanic [8], for which we prove the first bounds on worst-case running times – after roughly $O(n \log n)$ rounds all SINR constraints are nearly satisfied. When we try to satisfy each constraint exactly, however, convergence time is infinite. For this case, we design a novel framework for power control using regret learning algorithms and iterative discretization. While the exact convergence times must rely on a variety of parameters, we show that roughly a polynomial number of rounds suffices to make every link successful during at least a constant fraction of all previous rounds.

1 Introduction

A key ingredient to the operation of wireless networks is successful transmission in spite of interference and noise. Usually, a transmission is successful if the received signal strength is significantly stronger than the disturbance due to decayed signals of simultaneous transmissions and ambient noise. This condition is frequently expressed by the *signal-to-interference-plus-noise ratio (SINR)*. Over the last decade, a large amount of research work has studied the problem of throughput or *capacity maximization*, i.e., determining the maximum number of wireless transmissions that can be executed successfully in a network in parallel. Very recently, algorithms for capacity maximization are starting to receive interest also from an analytical and theoretical point of view. Most of the algorithms proposed and analyzed so far require a strong central authority managing the access of all devices to the spectrum. In addition, most works neglect *power control*, i.e., the ability of modern wireless devices to allow their transmission powers to be set by software. Power control has two main advantages. On the one hand, battery life can be increased by using only minimal powers that are necessary to guarantee reception. On the other hand, reduced transmission power causes

* This work has been supported by DFG through UMIC Research Centre, RWTH Aachen University, and grant Ho 3831/3-1.

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 637–649, 2011.

less interference, and thereby the throughput of a wireless network can increase significantly when using power control.

In this paper, we study spectrum access with power control in a network of wireless devices. We consider a network consisting of n *links*, i.e., sender/receiver pairs. Each sender attempts a successful transmission to its corresponding receiver using a transmission power. The chosen power has to be large enough to compensate the interference and ambient noise. In contrast, choosing a smaller transmission power is desirable as it results in less energy consumption. We investigate distributed algorithms to find transmission powers in order to make links successful as quickly possible. A standard assumption in the analysis of power-control problems of this kind is the existence of a solution, in which all transmissions are successful. For networks, in which this assumption does not hold, it is possible to combine the algorithms with approaches that solve the additional scheduling problem [13].

A simple and beautiful distributed algorithm for the power-control problem is the fixed-point iteration due to Foschini and Miljanic [8]. In each step every sender sets his power to the minimum power that was required to overcome interference and noise in the last round. It can be shown that powers converge to a feasible assignment, even if updates are not simultaneous [17]. The obtained power assignment is minimal in the sense that it is component-wise smaller than all other feasible assignments. It is known that this algorithm converges at a geometric rate [12] in a numerical sense. However, to the best of our knowledge, no convergence results in the sense of quantitative worst-case running times have been shown, neither for this nor for other distributed algorithms.

In this paper, we investigate two classes of distributed algorithms for power control and analyze the dependencies of running time and solution quality on several parameters of the structure of the instance. For example, our analysis of the Foschini-Miljanic fixed-point iteration in Section 2 uses the largest eigenvalues of the normalized gain matrix and the degree, to which the SINR constraint is fulfilled. Assuming that both these parameters are constant, our first main result (Theorem 1) shows that the FM iteration achieves polynomial convergence time. In particular, starting from all powers set to 0, for any constant $\delta > 0$ we reach in $O(n \log n)$ steps a power assignment that satisfies the SINR constraint of every link by a factor of at least $1 - \delta$.

It is easy to see that the FM iteration might never reach the fixed point if we start with all powers set to 0. Thus, if we insist on links satisfying the SINR constraint exactly, we get an infinite convergence time during which all links remain unsuccessful. To overcome this problem, in Section 3 we introduce a novel technique to compute power assignments employing distributed regret-learning algorithms. For algorithms that guarantee no swap regret [4], we can also guarantee convergence to the fixed point. The convergence properties rely on our analysis of the FM iteration and depend additionally on the position of the fixed point compared to noise vector and maximum allowed power. Assuming these ratios are bounded by a constant, our second main result (Theorem 5)

is that for every constant $\epsilon > 0$ after a polynomial number of steps, we can reach a situation in which every request has been successful with respect to the exact SINR constraint during at least a $(1-\epsilon)$ fraction of the previous steps. Our regret learning technique has the advantage of being applicable also to instances, in which not all links can be successful simultaneously.

1.1 Formal Problem Statement

We consider transmissions in general interference models based on SINR. If the sender of link j emits a signal at power p_j, then it is received by the receiver of link i with strength $g_{i,j} \cdot p_j$, where $g_{i,j}$ is called the *gain*. This includes the well-known special case of the *physical model*, where the gain depends polynomially on the distance between sender and receiver. The transmission within link i is successful if the SINR constraint

$$\frac{g_{i,i} \cdot p_i}{\sum_{j \neq i} g_{i,j} \cdot p_j + \nu} \geq \beta$$

is fulfilled, i.e., the SINR is above some threshold β. In the power control problem, our task is to compute a feasible power assignment such that the SINR constraint is fulfilled for each link. Furthermore, each link should use the minimal possible power. More formally, let the *normalized gain matrix* C be the $n \times n$ matrix defined by $C_{i,i} = 0$ for all $i \in [n]$ and $C_{i,j} = \beta g_{i,j}/g_{i,i}$ for $i \neq j$. The *normalized noise vector* η is defined by $\eta_i = \beta\nu/g_{i,i}$. The task is to find a vector p such that $p \geq C \cdot p + \eta$. Note that throughout this paper, we use $\leq$ and $\geq$ to denote the respective component-wise inequality.

The set of all feasible power assignments is a convex polytope. If it is non-empty, there is a unique vector p^* satisfying $p^* = C \cdot p^* + \eta$. In a full-knowledge, centralized setting, the optimal power vector p^* can simply be computed by solving the linear equation system $p^* = C \cdot p^* + \eta$. However, a wireless network consists of independent devices with distributed control and the matrix C is not known. We assume the devices can only make communication attempts at different powers and they receive feedback in the form of the achieved SINR or (in an advanced scenario) only whether the transmission has been successful or not.

For the scenario in which the achieved SINR is known after each transmission attempt, the FM iteration is $p^{(t+1)} = C \cdot p^{(t)} + \eta$, where the achieved and the target SINR are needed to run this iteration. Foschini and Miljanic showed that the sequence of vectors $p^{(t)}$ converges to p^* as t goes to infinity. One can show that the existence of $p^* \geq 0$ with $p^* \leq C \cdot p^*$ implies that the modulus of all eigenvalues of C must be strictly less than 1. In our analyses, we will refer to the maximal modulus of an eigenvalue as $\lambda_{\max}$.

For the regret-learning technique we assume that each link i uses a no-regret learning algorithm to select from a suitably defined discrete subset power values in an interval $[0, p_i^{\max}]$. So $p_i^{\max}$ is the maximal power level user i might

choose. Let Φ be a set of measurable functions such that each $\phi \in \Phi$ is a map $\phi\colon [0, p_i^{\max}] \to [0, p_i^{\max}]$. Given a sequence of power vectors $p^{(1)}, \ldots, p^{(T)}$, the Φ-regret link i encounters is

$$R_i^{\Phi}(T) = \sup_{\phi \in \Phi} \sum_{t=1}^{T} u_i(\phi(p_i^{(t)}), p_{-i}^{(t)}) - u_i(p_i^{(t)}, p_{-i}^{(t)}) \ ,$$

where u_i is a suitable utility function defined below. For our analyses, we consider two cases for the set Φ. For *external regret* each $\phi \in \Phi$ maps every power value to a single power p_ϕ. In contrast, to define *swap regret*, Φ contains all measurable functions. An infinite sequence is called no Φ-regret if $R_i^{\Phi}(T) = o(T)$. An algorithm that produces a no Φ-regret sequence is a no-Φ-regret algorithm.

We will see that under the utility functions we assume, there are distributed no-Φ-regret algorithms. It suffices for each user to only know after each transmission attempt if it has been successful.

1.2 Related Work

For about two decades, wireless networks with power control have been extensively studied. While at first research focused on engineering aspects, recently the topic has attracted interest among computer scientists. Algorithmic research so far focused on scheduling problems, where for a given network of senders and receivers the goal is to select a maximum feasible subset (the "independent set" problem) or to partition the links into the minimal number of feasible subsets (the "coloring" problem). Allowing a scheduling algorithm to choose powers has a significant impact on size and structure of links scheduled simultaneously in practice, as was shown by [15].

As a consequence, much effort has been put into finding algorithms for scheduling with power control. More recently, theoretical insights on the problem are starting to develop [9,14,6]. In particular, the independent set problem with power control has been shown to be NP-hard [1]. In most related work, however, the algorithmic power control problem is neglected by setting powers according to some "oblivious" schemes, e.g., using some polynomial depending on distance between sender and receiver. For independent set and coloring problems in metric spaces, usually the mean or square-root function achieves best results [7,10,11]. In addition, there are distributed approaches for the independent set problem using no-regret learning and uniform power [5,3]. In general, there are strong lower bounds when using oblivious power, as algorithms provide only trivial $\Omega(n)$-approximations in instances with n links of greatly varying length [7]. Power control can significantly improve this condition as exemplified by the recent centralized constant-factor approximation algorithm for the independent set problem by Kesselheim [13]. Being a centralized combination of scheduling and power control, this algorithm is rather of fundamental analytical interest and of minor relevance in heavily distributed systems like wireless networks.

Distributed algorithms exist especially for the power control problem without the scheduling aspect. In this case, a feasible power assignment is assumed to

exist that makes all links feasible. The goal is to find feasible power levels with minimal power consumption. This problem can be seen as a second step after scheduling and can be solved in a centralized fashion by solving a system of linear equations as noted above. Foschini and Miljanic [8] solved the problem using a simple iterative distributed algorithm. They showed that their iteration converges from each starting point to a fixed point if it exists. Extending this, Yates [17] proved convergence for a general class of iterative algorithms including also a variant for limited transmission powers and an iteration, in which users update powers asynchronously. Besides this, Huang and Yates [12] proved that all these algorithms converge geometrically. This means that the norm distance to the fixed point in time step t is given by a^t for some constant $a < 1$. However, this is only a bound on the convergence rate in the numerical sense and does not imply a bound on the time until links actually become successful.

In addition, more complex iterative schemes have been proposed in the literature. For a general survey about these algorithms and the power control problem, see Singh and Kumar [16].

2 Convergence Time of the Foschini-Miljanic Iteration

In this section, we analyze the convergence time of the Foschini-Miljanic iteration with $p^{(t)} = C \cdot p^{(t-1)} + \eta$. It will turn out to be helpful to consider the closed form variant

$$p^{(t)} = C^t \cdot p^{(0)} + \sum_{k=0}^{t-1} C^k \eta \ . \tag{1}$$

The iteration will never actually reach the fixed point, although getting arbitrarily close to it. However, during the iteration the SINR will converge to the threshold β. For each $\delta > 0$, there is some round T from which the SINR will never be below $(1-\delta)\beta$. Since maximizing the SINR is the main target, we strive to bound the time T until each transmission is "almost" feasible. That is, the SINR is above $(1-\delta)\beta$. For this purpose, it is sufficent that the current vector p satisfies $(1-\delta)p^* \leq p \leq (1+\delta)p^*$.

As a first result, we bound the convergence time in terms of n when starting from 0. We will see, that the time is independent of the values of p^* or η. The only parameter related to the instance is $\lambda_{\max}$ the maximum eigenvalue of C, which has to occur as for $\lambda_{\max} = 1$ no fixed point can exist at all. Assuming it to be constant, we show that after $O(n \log n)$ rounds we reach a power assignment that satisfies the SINR constraint of every link by a factor of at least $1 - \delta$.

Theorem 1. *Starting from $p^{(0)} = 0$ after $t \geq \frac{\log \delta}{\log \lambda_{\max}} \cdot n \cdot \log(3n)$ rounds, for all $p^{(t)}$ we have $(1-\delta)p^* \leq p^{(t)} \leq p^*$.*

Proof. Define the following auxiliary matrix $M = C^m$, where $m = \left\lceil \log \frac{1}{3n} / \log \lambda_{\max} \right\rceil$. As we can see, the modulus of all eigenvalues of M is bounded by $\frac{1}{3n}$. Furthermore, defining $\eta' = \sum_{k=0}^{m-1} C^k \eta$, we have $p^{(mt')} = \sum_{k=0}^{t'-1} M^k \eta'$. This also implies $p^* = \sum_{k=0}^{\infty} M^k \eta'$.

Now we consider the characteristical polynomial of M in expanded as well as in factored form:

$$\chi_M(x) = x^n + \sum_{i=0}^{n-1} a_i x^i = \prod_{j=1}^{n} (x - b_j) \ .$$

The (possibly complex) b_j values correspond to the eigenvalues. Therefore, we have $|b_j| \leq \frac{1}{3n}$ for all $j \in [n]$. The a_i values can be computed from the b_i values using

$$a_i = \sum_{\substack{S \subseteq [n] \\ |S| = n-i}} \prod_{j \in S} (-b_j) \ .$$

For the modulus this gives us

$$|a_i| \leq \sum_{\substack{S \subseteq [n] \\ |S| = n-i}} \prod_{j \in S} |b_j| \leq \binom{n}{n-i} \left(\frac{1}{3n}\right)^{n-i} \ .$$

This yields the following bound for their sum

$$\sum_{i=0}^{n-1} |a_i| \leq \sum_{k=0}^{n} \binom{n}{k} \left(\frac{1}{3n}\right)^k - 1 = \left(1 + \frac{1}{3n}\right)^n - 1 \leq \frac{1}{2} \ .$$

We now use the fact that $\chi_M(M) = 0$. This is, $M^n = -\sum_{i=0}^{n-1} a_i M^i$. Since all $M^k \eta'$ are non-negative, the following inequality holds

$$\begin{aligned} M^n p^* = \sum_{k=n}^{\infty} M^k \eta' &= \sum_{k=0}^{n-1} M^k \eta' \left(-\sum_{i=0}^{k} a_i\right) + \sum_{k=n}^{\infty} M^k \eta' \left(-\sum_{i=0}^{n-1} a_i\right) \\ &\leq \left(\sum_{i=0}^{n-1} |a_i|\right) \sum_{k=0}^{\infty} M^k \eta' \leq \frac{1}{2} \sum_{k=0}^{\infty} M^k \eta' = \frac{1}{2} p^* \ . \end{aligned}$$

Now consider $t \geq m \cdot n \cdot \log \frac{1}{\delta}$. We have $p^* - p^{(t)} = C^t p^* \leq M^{n \log \frac{1}{\delta}} p^* \leq \delta p^*$. This proves the theorem. □

One can see that this bound is almost tight as there are instances where $\Omega(n)$ rounds are needed. A simple example can be given as follows. Let C be defined by $C_{i+1,i} = 1$ for all i and all other entries 0, $\eta = (1, 0, \ldots, 0)$. The only eigenvalue of this matrix is 0. However, it takes n rounds until the 1 of the first component has propagated to the nth component and the fixed point is reached.

These instances require a certain structure in the values of p^* and η. As a second result, we would like to present a bound independent of n and for every possible starting point $p^{(0)}$ that takes p^* and η into consideration.

Theorem 2. *Starting from an arbitrary* $p^{(0)}$*, we have* $(1-\delta)p^* \leq p^{(t)} \leq (1+\delta)p^*$ *for all* $t \geq T$ *with*

$$T = \frac{\log \delta - \log \max_{i\in[n]} \left| \frac{p_i^{(0)}}{p_i^*} - 1 \right|}{\log \max_{i\in[n]} \left| 1 - \frac{\eta_i}{p_i^*} \right|} .$$

In order to prove this bound, it is useful to consider the weighted maximum norm, which has been used by Huang and Yates [12] before. Due to space constraints, this and some further proofs have to be omitted and can be found in the full version.

As assumed for the original FM iteration, we also focused on the case that powers can be chosen arbitrarily high so far. However, our bounds directly transfer to the case where there is some vector of maximum powers $p^{\max}$. In this setting, all powers are projected to the respective interval $[0, p_i^{\max}]$ in each round [17]. One can see that this can only have a positive effect on the convergence time since the resulting sequence is component-wise dominated by the sequence on unlimited powers.

3 Power Control via Regret Learning

The fixed-point approach analyzed above has some major drawbacks. For example, in many sequences – in particular the ones starting from 0 – the target SINR is never reached, because all powers increase in each step and therefore they are always too small. Another drawback is that, in order to adapt the power correctly, the currently achieved SINR has to be known. A last disadvantage to be mentioned is its lacking robustness. We assumed the fixed-point to exist. If for some reason this does not hold the iteration might end up where some powers are 0 or $p^{\max}$ even if the transmission is not successful.

In order to overcome these drawbacks, we design a different approach based on regret learning. Here, each link is a user, striving to have a successful transmission but using the least power possible. The user is assumed to decide which power $p_i \in [0, p_i^{\max}]$ to use based on an utility function. In particular, we assume that each user gets zero utility if the SINR is below the threshold and a positive one otherwise. However, this utility increases when using a smaller power. Formally, we assume utility functions of the following form:

$$u_i(p) = \begin{cases} f_i(p_i) & \text{if user } i \text{ is successful with } p_i \text{ against } p_{-i} \\ 0 & \text{otherwise} \end{cases}$$

where $f_i \colon [0, p_i^{\max}] \to [0, p_i^{\max}]$ is a continuous and strictly decreasing function for each $i \in [n]$. With p_{-i} we denote the powers chosen by all users but user i.

The utility functions have to be considered this way in order to capture the SINR constraint appropriately. On the one hand, each user's maximum is at the

point where the SINR condition is exactly met. On the other hand and at least as important, for each user having a successful transmission is always better than an unsuccessful one. This property cannot be modeled by a continuous function.

As a consequence, we can ensure that all no-swap-regret sequences converge to the optimal power vector p^*. Furthermore, the fraction of successful transmissions converges to 1. This is in contrast to the FM iteration, where starting from $p^{(0)} = 0$ all transmissions stay unsuccessful during the entire iteration.

As a first result, we can see that the only possibility that all links encounter zero swap regret is the sequence only consisting of p^*.

Theorem 3. *Given any sequence $p^{(1)}, \dots, p^{(T)}$ such that the swap regret for each user is 0, then $p^{(t)} = p^*$ for all t.*

Proof. For each user i let $\hat{p}_i = \max_{t\in[T]} p_i^{(t)}$. Now assume that $\hat{p} \le p^*$ does not hold. This means there is some user i for which $\hat{p}'_i := (C \cdot \hat{p} + \eta)_i < \hat{p}_i$. This user encounters non-zero swap regret because he could always use $\hat{p}'_i$ instead of $\hat{p}_i$. The user would still be successful in the same steps as before but get a higher utility each time he chose $\hat{p}_i$. Since this is a contradiction we have $\hat{p} \le p^*$.

Now let $\check{p}_i = \min_{t\in[T]} p_i^{(t)}$. Assume that $\check{p} \ge p^*$ does not hold. This implies that for some user $\check{p}_i < (C \cdot \check{p} + \eta)_i$. So user i is never successful when using power $\check{p}_i$ but would always be with p_i^* (since $\hat{p} \le p^*$). This is again a contradiction because user i would encounter a non-zero swap regret.

In total, we have that both $\hat{p} \le p^*$ and $\check{p} \ge p^*$, yielding $\hat{p} = \check{p} = p^*$. □

In contrast, zero external regret does not suffice. One can construct instances of the following kind. There are two "central" links, using power $p_i^{\max}$ in even rounds and $p_i^{\max}/2$ in odd rounds. Both have zero external regret because all allowed switching operations would yield the same power p' being used in even as well as in odd rounds, losing all utility from even rounds. However, by these two links using the maximal power, all $n-2$ remaining links are blocked. Although there might be a fixed point p^* at which all links are successful, the sequence can stay in a state where only 2 of the n links are successful at all.

4 Computing No-Swap-Regret Sequences

Observe that in our case, the users are given an infinite number of possible choices. Furthermore, in order to capture the SINR threshold appropriately the utility functions have to be modeled as non-continuous. Unfortunately, this yields standard no-swap-regret algorithms cannot be used in this scenario because, to the best of our knowledge, they require a finite number of actions [2] or convex action spaces and continuous and concave utility functions [18].

Luckily, no-regret sequences can be computed in a distributed way nevertheless. In order to achieve swap regret $\epsilon \cdot T$, we execute an arbitrary existing no-swap-regret algorithm on a finite subset of the available powers, which is chosen depending on ϵ. This finite subset is constructed by dividing the set of

powers into intervals of equal length and using the right borders as the input action set for the algorithm. Bounding the loss due to the restriction on the right borders, we can prove the following theorem.

Theorem 4. *Let Algorithm $\mathcal{A}$ be a no-swap-regret algorithm for finite action spaces achieving swap regret at most $O(T^a \cdot N^b)$ after T rounds in case of N actions for suitable constants $0 \leq a < 1$, $b \geq 0$. Then $\mathcal{A}$ can be used to compute a no-swap-regret sequence for power control, achieving swap-regret at most $O(T^{\frac{a+b}{1+b}})$ in T steps.*

In particular, if each link knows after each step which powers would have made it successful, we can use the $O(\sqrt{TN \log N})$ full-information algorithm proposed by Blum and Mansour [4] for the following result.

Corollary 1. *There is an algorithm achieving swap regret $O(T^{\frac{3}{4}})$.*

If each link only gets to know if the transmission at the actually chosen power suffices, it can nevertheless compute the value of the utility function for the chosen power. Therefore, in this case we are in the partial-feedback model. Here, we can apply the $O(N\sqrt{T \log N})$ algorithm by Blum and Mansour [4] to build the following algorithm.

Corollary 2. *There is an algorithm achieving swap regret $O(T^{\frac{4}{5}})$ that only needs to know if the transmissions carried out were successful.*

5 Convergence of No-Swap-Regret Sequences

So far, we have seen how to compute no-swap-regret sequences. In this section, the result is complemented by a quantitative analysis of a no-swap-regret sequence. We see that not only convergence to the optimal power vector p^* is guaranteed but also the fraction of rounds in which each link is successful converges to 1. In contrast, starting from certain vectors in the FM iteration no transmission is ever successful at all.

Theorem 5. *For every sequence $p^{(0)}, \cdots, p^{(T)}$ with swap regret at most $\epsilon \cdot T$ and for every $\delta > 0$ the fraction of steps in which user i sends successfully is at least*

$$Q \cdot \frac{f_i\left((1+\delta)p_i^*\right)}{f_i\left((1-\delta)p_i^*\right)} - \frac{\epsilon}{f_i\left((1-\delta)p_i^*\right)} \quad ,$$

where Q denotes the fraction of rounds in which a power vector p with $(1-\delta)p^* \leq p \leq (1+\delta)p^*$ is chosen.

Given a sequence with swap regret at most $\epsilon \cdot T$, Theorem 5 gives a lower bound for the number of steps in which a user can send successfully. The bound depends on the utility function and the fraction of rounds in which a power vector between $(1-\delta)p^*$ and $(1+\delta)p^*$ is chosen. For this we give a bound in Lemma 1 and Lemma 3 later on. Altogether Theorem 5, Lemma 1, and Lemma 3 yield a bound converging to 1 as the swap regret per step approaches 0.

In order to prove this theorem, we will switch to a more convenient notation from game theory, namely correlated equilibria. Similar to a mixed Nash equilibrium, an ϵ-correlated equilibrium is a probability distribution over strategy vectors (in our case power vectors) such that no user can unilaterally increase his expected utility by more than ϵ. In contrast to mixed Nash equilibria the choices of the different users do not need to be independent. Formally, an ϵ-correlated equilibrium is defined as follows.

Definition 1. *An ϵ-correlated equilibrium is a joint probability distribution π over the set of power vectors $P_1 \times \cdots \times P_n$, where $P_i = [0, p_i^{\max}]$, such that for any user i and measurable function $\phi_i \colon P_i \to P_i$, we have*

$$\mathbf{E}_{s\sim\pi}\left[u_i(\phi_i(p_i), p_{-i})\right] - \mathbf{E}_{s\sim\pi}\left[u_i(p_i, p_{-i})\right] \leq \epsilon \ .$$

This is, in an ϵ-correlated equilibrium no user can increase his expected utility by operations such as "each time π says I play A, I play B instead". These kinds of operations are exactly the ones considered in the definition of no-swap-regret sequences. Therefore each sequence $p^{(1)}, \ldots, p^{(T)}$ of swap regret at most R corresponds to an R/T-correlated equilibrium.

Using this notion, we can rewrite Theorem 5 to the following proposition.

Proposition 1. *For every ϵ-correlated equilibrium π and for every $\delta > 0$ the probability that user i sends successfully is at least*

$$\mathbf{Pr}_{p\sim\pi}\left[(1-\delta)p^* \leq p \leq (1+\delta)p^*\right] \frac{f_i\left((1+\delta)p_i^*\right)}{f_i\left((1-\delta)p_i^*\right)} - \frac{\epsilon}{f_i\left((1-\delta)p_i^*\right)} \ .$$

To prove the proposition, we consider the switching operation where powers in the interval $[(1-\delta)p_i^*, (1+\delta)p_i^*]$ are exchanged by $(1+\delta)p_i^*$. We then use the fact that after switching the user would always be successful unless the other users choose powers $p_{-i} > (1+\delta)p_{-i}^*$. Since π is an ϵ-correlated equilibrium, this operation can increase the expected utility by at most ϵ. Therefore the expected utility before the switch and this way also the success probablity cannot be too small.

It remains to bound the probability $\mathbf{Pr}_{p\sim\pi}\left[(1-\delta)p^* \leq p \leq (1+\delta)p^*\right]$. For this purpose, we bound the probability mass of states p with $p \not\leq (1+\delta)p^*$ in Lemma 1 and of the ones with $p \not\geq (1-\delta)p^*$ in Lemma 3. This way, we get the desired bound by

$$\begin{aligned} &\mathbf{Pr}_{p\sim\pi}\left[(1-\delta)p^* \leq p \leq (1+\delta)p^*\right] \\ &= 1 - \mathbf{Pr}_{p\sim\pi}\left[p \not\geq (1-\delta)p^*\right] - \mathbf{Pr}_{p\sim\pi}\left[p \not\leq (1+\delta)p^*\right] \ . \end{aligned}$$

The general proof ideas work as follows. In order to bound $\mathbf{Pr}_{p\sim\pi}\left[p \not\leq (1+\delta)p^*\right]$, we consider which probability mass can at most lie on vectors p such that for some user i, we have $p_i > \left(C \cdot p^{\max} + (1+\delta/2)\cdot\eta\right)_i$. This probability mass is bounded, because user i could instead always use power $\left(C \cdot p^{\max} + \eta\right)_i$, as this is the maximum power needed to compensate the interference in the case that

$p_{-i} = p_{-i}^{\max}$. We then proceed in a similar way always using the bound obtained before until we reach a point component-wise smaller than $(1+\delta)p^*$. The bound on $\mathbf{Pr}_{p\sim\pi}[p \not\geq (1-\delta)p^*]$ works in a similar way.

To see which probability mass lies on states by a factor δ away from the fixed point p^*, we will now consider how much probability mass can at most lie on states $p \not\leq (1+\delta)p^*$. Afterwards, we will do the same for $p \not\geq (1-\delta)p^*$.

Lemma 1. *Let π be an ϵ-correlated equilibrium for some $\epsilon \geq 0$. Then for all $\delta > 0$, we can bound the probability that $p \not\leq (1+\delta)p^*$ is chosen by*

$$\mathbf{Pr}_{p\sim\pi}[p \not\leq (1+\delta)p^*] \leq \epsilon \left(\frac{n}{\delta} \max_{i\in[n]} \frac{2}{s_i \eta_i} + 2\right)^{T+1}$$

$$\text{where } T = \frac{\log\frac{\delta}{4} - \log\max_{i\in[n]} \left|\frac{p_i^{max}}{(1+\frac{\delta}{2})p_i^*}\right|}{\log\max_{i\in[n]} \left|1 - \frac{\eta_i}{p_i^*}\right|} \ ,$$

where s_i denotes the minimal absolute value of the difference quotient of f_i at any point p_i and $p_i + \frac{\delta}{2}\eta_i$.

The probability that vectors below $(1-\delta)\,p^*$ are chosen can be bounded in similiar ways. For this, we define $r = \min_i r_i$, and r_i is a lower bound on the utility of user i at $(1+\delta)p^*$, i.e., $r \leq \min_{i\in[n]} f_i((1+\delta)p_i^*)$.

Lemma 2. *Let π be an ϵ-correlated equilibrium for some $\epsilon \geq 0$. Then for all $\delta > 0$, we can bound the probability that $p \not\geq (1-\delta)p^*$ is chosen by*

$$\mathbf{Pr}_{p\sim\pi}[p \not\geq (1-\delta)p^*] \leq \left(\frac{\epsilon}{1-r} + \mathbf{Pr}_{p\sim\pi}[p \not\leq (1+\delta)p^*]\right)\left(\frac{n}{r}\right)^{T'+1}$$

$$\text{where } T' = \frac{\log\delta}{\log\max_{i\in[n]} \left|1 - \frac{\eta_i}{p_i^*}\right|} \ .$$

Having already found a bound for $\mathbf{Pr}_{p\sim\pi}[p \not\leq (1+\delta)p^*]$ in Lemma 1, we can directly conclude the following.

Lemma 3. *Given an ϵ-correlated equilibrium and $u_i(p^{max}) \geq r = \frac{1}{2}$ for all $i \in [n]$. Then for every $\delta > 0$ the probability that a vector $p \not\geq (1-\delta)p^*$ is chosen is at most*

$$\mathbf{Pr}_{p\sim\pi}[p \not\geq (1-\delta)p^*] \leq \epsilon \left(2 + \left(\frac{n}{\delta} \max_{i\in[n]} \frac{2}{s_i \eta_i} + 2\right)^{T+1}\right)(2n)^{T'+1}$$

$$\text{with } T' = \frac{\log\delta}{\log\max_{i\in[n]} \left|1 - \frac{\eta_i}{p_i^*}\right|} \text{ and } T = \frac{\log\frac{\delta}{4} - \log\max_{i\in[n]} \left|\frac{p_i^{max}}{(1+\frac{\delta}{2})p_i^*}\right|}{\log\max_{i\in[n]} \left|1 - \frac{\eta_i}{p_i^*}\right|} \ .$$

Combining Lemma 1 and 3, we get an upper bound on $\mathbf{Pr}_{p\sim\pi}\left[(1-\delta)p^* \leq p \leq (1+\delta)p^*\right]$. For appropriately chosen δ, this bound and Proposition 1 yield that the success probability converges to 1 as ϵ approaches 0. This also yields that in each no-swap-regret sequence for each user the limit of the fraction of successful steps is 1. Furthermore, the chosen powers also have to converge to p^*.

6 Discussion and Open Problems

In this paper, we studied two distributed power control algorithms. We obtained the first quantitative bounds on how long it takes in the FM iteration until the SINR is close to its target value. Furthermore a novel approach based on regret learning was presented. It overcomes some major drawbacks of the FM iteration. It is robust against users that deviate from the protocol and it still converges in a partial-information model, where the achieved SINR is not known. For no-swap-regret algorithms the convergence of the regret-learning approach is guaranteed.

Considering general no-swap-regret sequences is only a weak assumption and therefore the obtained bounds are not as good as the ones of the FM iteration. This yields a perspective for possible future work. An algorithm particular for power control could be designed based on the regret-learning approach presented in this paper.

Another aspect to be considered in future work could be discretization of the power levels. The standard assumption is that users can choose arbitrary real numbers as powers. In realistic devices this assumption might not be applicable. To the best of our knowledge, the additional challenges arising in this case have not been considered so far.

References

1. Andrews, M., Dinitz, M.: Maximizing capacity in arbitrary wireless networks in the sinr model: Complexity and game theory. In: Proceedings of the 28th Conference of the IEEE Communications Society, INFOCOM (2009)
2. Arora, S., Hazan, E., Kale, S.: The multiplicative weights update method: a meta algorithm and applications (2005) (manuscript)
3. Asgeirsson, E.I., Mitra, P.: On a game theoretic approach to capacity maximization in wireless networks. In: Proceedings of the 30th Conference of the IEEE Communications Society, INFOCOM (2011)
4. Blum, A., Mansour, Y.: From external to internal regret. J. Mach. Learn. Res. 8, 1307–1324 (2007)
5. Dinitz, M.: Distributed algorithms for approximating wireless network capacity. In: Proceedings of the 29th Conference of the IEEE Communications Society (INFOCOM), pp. 1397–1405 (2010)
6. Fanghänel, A., Geulen, S., Hoefer, M., Vöcking, B.: Online capacity maximization in wireless networks. In: Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures, pp. 92–99 (2010)

7. Fanghänel, A., Kesselheim, T., Räcke, H., Vöcking, B.: Oblivious interference scheduling. In: Proceedings of the 28th ACM Symposium on Principles of Distributed Computing (PODC), pp. 220–229 (2009)
8. Foschini, G.J., Miljanic, Z.: A simple distributed autonomous power control algorithm and its convergence. IEEE Transactions on Vehicular Technology 42(4), 641–646 (1992)
9. Goussevskaia, O., Wattenhofer, R., Halldórsson, M.M., Welzl, E.: Capacity of arbitrary wireless networks. In: Proceedings of the 28th Conference of the IEEE Communications Society (INFOCOM), pp. 1872–1880 (2009)
10. Halldórsson, M.M.: Wireless scheduling with power control. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 361–372. Springer, Heidelberg (2009)
11. Halldórsson, M.M., Mitra, P.: Wireless capacity with oblivious power in general metrics. In: Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1538–1548 (2011)
12. Huang, C.Y., Yates, R.D.: Rate of convergence for minimum power assignment algorithms in cellular radio systems. Wireless Networks 4(4), 223–231 (1998)
13. Kesselheim, T.: A constant-factor approximation for wireless capacity maximization with power control in the SINR model. In: Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1549–1559 (2011)
14. Kesselheim, T., Vöcking, B.: Distributed contention resolution in wireless networks. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 163–178. Springer, Heidelberg (2010)
15. Moscibroda, T., Wattenhofer, R.: The complexity of connectivity in wireless networks. In: Proceedings of the 25th Conference of the IEEE Communications Society (INFOCOM), pp. 1–13 (2006)
16. Singh, V., Kumar, K.: Literature survey on power control algorithms for mobile adhoc network. Wireless Personal Communications, 1–7 (2010), doi:10.1007/s11277-010-9967-x
17. Yates, R.D.: A framework for uplink power control in cellular radio systems. IEEE Journal on Selected Areas in Communications 13(7), 1341–1347 (1995)
18. Zinkevich, M.: Online convex programming and generalized infinitesimal gradient ascent. In: Proceedings of the 20th International Conference on Machine Learning, pp. 928–936 (2003)

A New Approach for Analyzing Convergence Algorithms for Mobile Robots*

Andreas Cord-Landwehr, Bastian Degener, Matthias Fischer, Martina Hüllmann, Barbara Kempkes, Alexander Klaas, Peter Kling, Sven Kurras, Marcus Märtens, Friedhelm Meyer auf der Heide, Christoph Raupach, Kamil Swierkot, Daniel Warner, Christoph Weddemann, and Daniel Wonisch

Heinz Nixdorf Institute & Department of Computer Science
University of Paderborn, Germany

Abstract. Given a set of n mobile robots in the d-dimensional Euclidean space, the goal is to let them converge to a single not predefined point. The challenge is that the robots are limited in their capabilities. Robots can, upon activation, compute the positions of all other robots using an individual affine coordinate system. The robots are indistinguishable, oblivious and may have different affine coordinate systems. A very general discrete time model assumes that robots are activated in arbitrary order. Further, the computation of a new target point may happen much earlier than the movement, so that the movement is based on outdated information about other robot's positions. Time is measured as the number of rounds, where a round ends as soon as each robot has moved at least once. In [6], the Center of Gravity is considered as target function, convergence was proven, and the number of rounds needed for halving the diameter of the convex hull of the robot's positions was shown to be $\mathcal{O}(n^2)$ and $\Omega(n)$. We present an easy-to-check property of target functions that guarantee convergence and yields upper time bounds. This property intuitively says that when a robot computes a new target point, this point is significantly within the current axes aligned minimal box containing all robots. This property holds, e.g., for the above-mentioned target function, and improves the above $\mathcal{O}(n^2)$ to an asymptotically optimal $\mathcal{O}(n)$ upper bound. Our technique also yields a constant time bound for a target function that requires all robots having identical coordinate axes.

1 Introduction

Over the last decade there has been a growing interest in problems related to the creation of formations by autonomous robots. Given n robots in the Euclidean space $\mathbb{R}^d$, one of the most intuitive problems is to consider their goal to converge

* Partially supported by the EU within FP7-ICT-2007-1 under contract no. 215270 (FRONTS) and DFG-project "Smart Teams" within the SPP 1183 "Organic Computing" and International Graduate School "Dynamic Intelligent Systems"

L. Aceto, M. Henzinger, and J. Sgall (Eds.): ICALP 2011, Part II, LNCS 6756, pp. 650–661, 2011.

to a common point. This convergence problem is widely examined under various assumptions about the capabilities of the robots and the time model. In this paper, we assume the Look-Compute-Move model (LCM model) [6,15,19,18]. Therein, each robot performs a sequence of *cycles*, where each cycle consists of three distinct instantaneous *operations*, namely Look (observe the current positions of all robots), Compute (use some *target function* to compute a new target point from the observed robot positions) and Move (move towards the target point) in this order. We use the most general time model (the *asynchronous model*), which does not assume any coupling of the LCM steps of different robots. As a consequence of this asynchronism, many operations of other robots can take place between the Look- and the Move-operation of the same cycle of a robot so that the current positions may have substantially changed, compared to the formerly observed positions that were used as input to the target function. Runtime is measured in terms of *rounds*. A round ends as soon as all robots have executed at least one complete LCM cycle since the end of the previous round.

This convergence problem is widely considered in literature. The main known results use the robots' Center of Gravity (CoG) as target function. We subclassify the models by their movement details: In the LCM_∞ model, the robots always exactly reach their target points during the Move operation. In the LCM_S model, the movement towards the target might alternatively stop but covers at least some fixed distance $S > 0$. Cohen and Peleg [6] showed upper bounds of $\mathcal{O}(n^2)$ for the LCM_∞ model, and $\mathcal{O}(n^2 + n\frac{h}{S})$ for the LCM_S model, bounding the number of rounds needed for halving the diameter of the set containing the robot positions and target points, assuming an initial diameter h. In addition, they present an $\Omega(n)$ lower bound for both models.

1.1 Our Contribution

We present a general upper bound for a parametrized class of target functions. The key characteristic of the robots' target function, which we introduce and analyze in this paper, is the δ-*inner property* for a parameter $\delta \in (0, \frac{1}{2}]$: For a time step t, consider the minimal axes aligned box $B(t) := \text{minbox}(P(t))$ containing all current robot positions. The δ-inner of $B(t)$ is the axes aligned box included in $B(t)$ with distance $\delta \cdot D_k(t)$ to the two boundaries of $B(t)$ in dimension k, $1 \leq k \leq d$. $D_1(t), \ldots, D_k(t)$ denote the side lengths of $B(t)$.

Definition 1. *A robot's target function has the* δ-inner property, *if the target point computed by the robot at time* t *always lies in the* δ*-inner of* $\text{minbox}(P(t))$.

We provide proofs of convergence and runtime bounds depending on δ for all target functions that fulfill the δ-inner property. For different robots and even different time steps, the target functions may be different. For the LCM_∞ model we prove an upper bound of $\mathcal{O}(\frac{1}{\delta})$ and for the LCM_S model of $\mathcal{O}(\frac{1}{\delta} + \frac{D_{max}}{S\delta})$, where D_{max} denotes the maximum diameter of the set of all robot positions and target points. Since the Center of Gravity has the $1/n$-inner property, this yields an optimal $\mathcal{O}(n)$ upper bound for the LCM_∞ model, and an improved $\mathcal{O}(n+n\frac{h}{S})$ upper bound for the LCM_S model, with h as the initial diameter. As

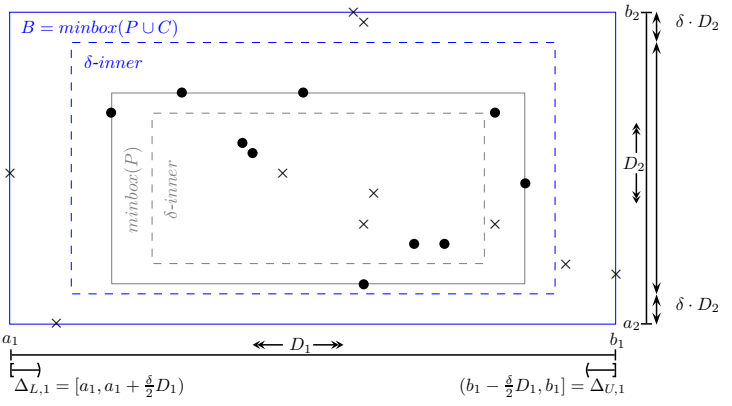


Fig. 1. Exemplary scenario at a time t: • marks robot positions, × target points

a further simple application, we present the Center-of-Minbox target function, prove that it has the 1/2-inner property, and conclude a constant runtime bound. Note that this strategy assumes equally aligned coordinate axes at all robots (however, orientation and order may vary).

1.2 Ours and Other's Models

For our analysis we use the initially mentioned LCM model in which each robot operates individually by repeating the same three step algorithm, consisting of a Look-, a Compute-, and a Move-operation (LCM) like in [6]. The time model for our analysis is the asynchronous time model ($\mathcal{ASYNC}$) [4,6], which allows the different operations of a robot to be arbitrarily nested with other robots' operations. That is, at time $t \in \mathbb{N}$, each robot out of an arbitrary subset performs an operation (Look, Compute, or Move), where different robots may perform different operations at the same time. This model also includes the more specific semi-synchronous ($\mathcal{SSYNC}$) [6,18] and full-synchronous ($\mathcal{FSYNC}$) [19] time models in which some or all robots execute equal operations synchronously. As an optional restriction to LCM, one can terminate a robot's motion before it actually reaches its target point in the Move-operation. For this, we introduce a sub-classification of the LCM model into two LCM movement models:

- LCM_∞: Each robot reaches the target point at the end of its Move-operation. (This is known as *undisturbed-motion* in the $\mathcal{ASYNC}$ time model in [6].)
- LCM_S: Each robot moves on a straight line towards its target point. It either reaches this point or moves at least a constant distance S. (This is known as restriction (A2) in [3] and asynchronous *sudden-stop* model in [6].)

We group the robot activations into cycles and rounds: If a robot completes a Look-, Compute-, and Move-operation we call this a *cycle*. We assume that two consecutive cycles of a robot take place within finite time. (Note that this

also implies that no robot crashes.) Starting at any time we call a sequence of cycles a *round* at the earliest point in time when all robots have performed at least one complete cycle. We count the number of rounds to state the runtime bounds. Note that, as long as the target function solely depends on the data observed in the previous LOOK operation, it does not make any difference when exactly the COMPUTE operation takes place. We will therefore usually assume that the COMPUTE-operation is performed directly after the LOOK-operation, and we will refer to this combined operation by the *update* of a robot's target point. This unification simplifies the terminology and notation, but it does not alter the behavior of the algorithm and yields the same results.

Notations. For a robot $i \in R$ from the set of robots $R := \{1, \ldots, n\}$ and a time $t \in \mathbb{N}$, we denote its position in the d-dimensional Euclidean space by $p_i(t) \in \mathbb{R}^d$ and its target point from its last COMPUTE-operation by $c_i(t) \in \mathbb{R}^d$, both after applying any current operations. Further, the set of all robots' positions at time t is denoted by $P(t)$ and the set of all target points by $C(t)$.

In our analysis, we fix some arbitrary coordinate system. For a given finite point set $A \subset \mathbb{R}^d$, we denote by minbox(A) the minimum bounding d-dimensional hyperbox that is axis-parallel to our chosen coordinate system and contains A. Note that the convex hull conv(A) is always contained inside the usually significantly larger minbox. In particular, we are interested in minbox$(P(t) \cup C(t))$. Its side length in dimension k is $D_k(t)$, $1 \leq k \leq d$.

The crucial notion for our following analysis is the term of the δ-*inner* of a minbox for some fixed $\delta \in (0, \frac{1}{2}]$. Here, for a minbox M its δ-inner denotes the box' subset that is retrieved by scaling M down to factor $(1 - 2\delta)$ with fixed center. This also defines the δ-*border* as the boundary region that is created by subtracting the δ-inner from M. In the projection onto some coordinate axis k, this yields a *lower* and an *upper* δ-border of length $\delta \cdot D_k$, each.

Recall that a target function of a robot i has the δ-inner property if a target computed by i at time t lies in the δ-inner of minbox$(P(t))$. Our analysis will be based on the $\delta/2$-inner of minbox$(P(t) \cup C(t))$. We use the following notations for the outer half of the δ-border, corresponding to the individual dimensions. For minbox$(P(t) \cup C(t)) =: [a_1(t), b_1(t)] \times \ldots \times [a_d(t), b_d(t)]$ we denote the *lower* $\delta/2$-*border of dimension* k by $\Delta_{L,k}(t) = \left[a_k(t),\ a_k(t) + \frac{\delta}{2} \cdot D_k(t)\right)$ and analogously the *upper* $\delta/2$-*border of dimension* k by $\Delta_{U,k}(t) = \left(b_k(t) - \frac{\delta}{2} \cdot b_k(t),\ b_k(t)\right]$. We call a subset of $\mathbb{R}^d$ to be *empty* if it contains neither a robot nor any target point. For any time t, $R_{\text{LOOK}}(t)$ consists of the robots that are currently performing their LOOK-operations, alike $R_{\text{MOVE}}(t)$ the robots performing their MOVE-operations.

1.3 Related Work

Our work closes the convergence speed gap of the Center of Gravity algorithm by Cohen and Peleg [5,6]. There, the authors analyze an algorithm where each robot computes the Center of Gravity of all robot positions as its target point. This algorithm is analyzed in the asynchronous time model ($\mathcal{ASYNC}$) and results are upper bounds of $\mathcal{O}(n^2)$ rounds in the LCM_∞ model and of $\mathcal{O}(n^2 + \frac{nh}{S})$ rounds

in the LCM_S model, with S as the minimal step length and h as the initial diameter of the set of robots. Counted is the number of rounds until the convex hull of the robot positions is halved in each dimension. This is one of the rare examples with stated runtime bounds for a formation problem.

An also well studied but harder variant of the convergence problem is the gathering problem. There, the robots do not only need to converge to a point, but actually reach this point. For the gathering as well as for the convergence problem, most available work investigates which robot capabilities are crucial in which time model to achieve convergence or gathering [2,6,7] or that are handy to simplify a problem's instance ([12] uses a common coordinate system). Those capabilities are for example memory in contrast to obliviousness, orientation to have common coordinate systems, identities to enable robot distinctions, or multiplicity-detection to discover whether several robots occupy a used position.

Concerning results without runtime bounds, [11] states that anonymous robots, which are disoriented, oblivious, and cannot communicate with each other can gather in $\mathcal{SSYNC}$ if and only if the number of robots is odd. The problem of having inaccurate compasses as a challenge for two robots is discussed in [17]. [14] investigates local compasses that vary over time. Another focus are negative results for gathering, e.g. see [16]. If at least one robot behaves maliciously, gathering is only possible with at least three robots in total [1]. In [9] the authors equip robots with an extent and face the challenge that robots' views can be blocked by other robots. All these results are stated without runtime bounds.

A recent result with stated runtime bounds for the gathering problem with robots that have an extent was given in [8]. For gathering with limited visibility, a recent algorithm with runtime bounds is due to Degener et al. [10]. Further, the authors in [13] point out that having an algorithm without any assumptions on the robots leads to an exponential lower bound for randomized algorithms for gathering; also they propose a linear time gathering algorithm on the base of multiplicity detection. However, in contrast to our work, randomization is used.

1.4 Organization of the Paper

At first we present insights into the progress achieved during one round by using a δ-inner target function. These insights bring out runtime bounds for LCM_∞ and LCM_S in Sections 2.1 and 2.2. Applications in Sections 3 and 4 then yield runtime bounds for the Center of Gravity and the Center of Minbox algorithms.

2 Convergence Speed of δ-inner Target Functions

By *convergence speed* we denote the number of rounds that are needed for halving the diameter along each dimension of the minbox around all robot positions and target points. In this section, we present bounds for the worst-case convergence speed for algorithms that fulfill the δ-inner property. Here, we handle each LCM movement model separately, since it turns out that LCM_∞ can directly be managed by our technique, whereas LCM_S needs more geometric arguments.

Lemma 1 (Monotonicity). *Let* $M(t) := \text{minbox}(P(t) \cup C(t))$ *and* $t < t'$ *be two points in time. Then it holds:* $M(t) \supseteq M(t')$

Proof. Let t' be the first time when some robot i becomes active after time t. If i computes a new target point c, this c lies in the δ-inner of $\text{minbox}(P(t))$, which is included in $M(t)$. If i moves, then it moves towards $c_i(t) \in C(t)$. By convexity of a minbox, i never leaves $M(t)$. □

The following lemma carries out the essential idea for our proof. We fix a dimension and analyze the projection of the robot positions and target points onto it. Whenever there is some robot calculating a target point in one of the two $\delta/2$-border segments for this dimension, and later there is some further robot calculating a target point in the other $\delta/2$-border segment, then the previous segment can no longer contain any robot or target point, i.e., it is empty; this is the previously mentioned migration process. Let t_0 be a fixed time, we define $\Delta_L := \Delta_{L,k}(t_0)$ and $\Delta_U := \Delta_{U,k}(t_0)$. Remember that this corresponds to the respective outer half of the δ-border.

Lemma 2 (Migration). *Let* $R = \{1, \ldots, n\}$ *be a set of robots in the asynchronous* LCM_∞ *model with* δ*-inner target functions. For a fixed dimension* k *consider the scenario's projection onto the* k*'th coordinate axis. Let* $t_0 \leq t_L \leq t_U$ *be points in time and assume* $\text{diam}(\text{minbox}(P(t_0) \cup C(t_0))) > 0$. *If there is*

1. *a robot* $i \in R$ *that updates its target point at time* t_L *to* $c_i(t_L) \in \Delta_L$ *and*
2. *a robot* $j \in R$ *that updates its target point at time* t_U *to* $c_j(t_U) \in \Delta_U$

then Δ_L *is empty at time* t_U. *The same holds for* $\Delta_{L,k}(t_0)$ *and* $\Delta_{U,k}(t_0)$ *swapped.*

Proof. Consider the projection of all robot positions and target points at time t_0 onto coordinate axis k. Their smallest enclosing interval at this axis is denoted by $[a, b]$. Further, we define H_L as the lower half and H_U as the upper half of this interval, both excluding the center position.

Since i updates its target point at time t_L to $c_i(t_L) \in \Delta_L$, we get by the δ-inner property that the δ-inner of all current robot positions at time t_L must intersect Δ_L. As the robots cannot leave $[a, b]$ by geometric argument all robots are in H_L at time t_L. Alike, at time t_U all robots are in H_U and by $H_L \cap H_U = \emptyset$ we get $t_U > t_L$.

Our claim is that after period $[t_L, t_U]$ the interval Δ_L contains neither any robot position nor any target point. For any time t we define the set $\mathcal{L}(t) := \{i \in R \mid c_i(t) \in \Delta_L\}$. As $\Delta_L \cap H_U = \emptyset$ and hence no robot is in Δ_L at time t_U, it remains to show that $\mathcal{L}(t_U)$ is empty.

Let $\hat{t} \in (t_L, t_U]$ be the earliest point in time such that for all $t \in [\hat{t}, t_U]$ at least one robot is in H_U. (Note that t_U fulfills this constraint and hence $\hat{t}$ is well-defined.) We have that in period $[\hat{t}, t_U]$ the δ-inner of the robot positions is disjoint to Δ_L and hence no robot's update at any time $t \in [\hat{t}, t_U]$ can result in a target point in Δ_L. Thus it is for all $t \in [\hat{t}, t_U] : \mathcal{L}(t) \subseteq \mathcal{L}(\hat{t})$.

Finally, consider $\mathcal{L}(\hat{t})$: Since by choice $\hat{t}$ is minimal, no robot is in H_U at time $\hat{t} - 1$. Further, a robot that enters H_U at time $\hat{t}$ must come from outside of H_U

and has a target point $c(\hat{t})$ in H_U. So it cannot be contained in $\mathcal{L}(t)$. By this we characterize all robots $r \in \mathcal{L}(\hat{t})$ to have $p_r(\hat{t}) \notin H_U$ and $c_r(\hat{t}) \in \Delta_L$.

Since each robot from $\mathcal{L}(t_U)$ moves to H_U in period $(\hat{t}, t_U]$—which is the so called *migration process*—each robot from $\mathcal{L}(\hat{t})$ recalculates its target point to a point in H_U and by this to a point outside of Δ_L. With the former monotony argument of $\mathcal{L}(t)$ we get $\mathcal{L}(t_U) = \emptyset$ and thus, Δ_L is empty at time t_U. □

We conclude the following Main Lemma which shows that the minbox of robot positions and target points shrinks in every round. It says that after one round, either one of the $\delta/2$-borders is empty (in each dimension), or the robots from one of the borders compute a target point in the $\delta/2$-inner of the box and move towards it. Both cases reduce the size of the minbox.

Main Lemma 1. *In the asynchronous* LCM *model, after any round starting at time t_0 it holds for any dimension k with $D_k(t_0) > 0$: Either Δ_L or Δ_U is empty or, for at least one of both intervals, no target point is computed in this interval during the entire round.*

Proof. If for one of the two intervals Δ_L and Δ_U it holds that no target point is computed in the interval during the entire round, the lemma follows. Otherwise, we can apply Lemma 2: At the time where the target point is computed in the second interval, the first interval is empty. Because of the monotonicity of the minbox of target points and robot positions (Lemma 1), no target point will ever be computed in this interval again. □

Fact 1. *Let a function $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$ satisfy the following recursion for some fixed $c \in (0, 1]$: $f(k+1) \leq (1-c) \cdot f(k)$, for $k > 0$ and arbitrary $f(0) \in \mathbb{R}_{\geq 0}$. Then, for all $k \in \mathbb{N} : f(k+\alpha) \leq \frac{1}{2} \cdot f(k)$ for $\alpha := \left\lceil \frac{1}{c \cdot \log_2(e)} \right\rceil \in \mathcal{O}(\frac{1}{c})$.*

Our strategy is to show the recursion $D_k(t_{end}) \leq (1-c) \cdot D_k(t_{start})$ for each round $[t_{start}, t_{end}]$ and $c \in (0, 1]$ separately in each dimension k, from which we can derive the convergence speed $\mathcal{O}(\frac{1}{c})$ for halving along this dimension.

2.1 Convergence Speed in the Asynchronous LCM_∞ Model

We first consider the number of rounds needed until one of the minbox' $\delta/2$-borders is empty. Second, by this number we bound the total convergence time.

Lemma 3. *In the asynchronous LCM_∞ model with δ-inner target functions, after any round in the time interval $[t_{start}, t_{end}]$ it holds for any dimension k with $D_k(t_{start}) > 0$ that $\Delta_L := \Delta_{L,k}(t_{start})$ or $\Delta_U := \Delta_{U,k}(t_{start})$ is empty.*

Proof. Assume Δ_L and Δ_U both being not empty at time t_{end}. The Main Lemma gives us that in one of both intervals, w.l.o.g. assume in Δ_L, no target point is calculated during the complete round. Since all robots perform at least one cycle, they calculate a target point outside of Δ_L and then have to leave Δ_L to reach it in the subsequent MOVE-operation. Further, they cannot re-enter Δ_L until the end of the round without recalculating a target point in Δ_L. Hence, Δ_L is empty at the end of the round. □

Theorem 1. *In the asynchronous LCM_∞ model with δ-inner target functions, after each round the diameter of the minbox around all robot positions and target points is halved in each dimension separately after $\mathcal{O}(1/\delta)$ rounds.*

Proof. Fix any dimension k and any round at time interval $[t_{start}, t_{end}]$. By Lemma 3, Δ_L or Δ_U is empty at time t_{end}. Since it is $|\Delta_L| = |\Delta_U| = \frac{\delta}{2} \cdot D_k(t_{start})$ we get: $D_k(t_{end}) \leq (1 - \frac{\delta}{2}) \cdot D_k(t_{start})$. According to Fact 1, D_k is halved after $\mathcal{O}(1/\delta)$ rounds. □

Remark 1. The given worst-case runtime bound from Theorem 1 is even tight: By application of a similar construction as in [6] for δ-inner target functions, we can get scenarios for which $\Omega(1/\delta)$ rounds are needed for halving the diameter in the worst-case. This leads to the tight worst-case runtime bound of $\Theta(1/\delta)$ in the LCM_∞ model.

2.2 Convergence Speed in Asynchronous LCM_S Model

A similar construction can be used to prove the convergence speed in the LCM_S model. Here, let $S > 0$ denote the fixed minimum distance that any robot moves during its MOVE-operation, if it does not reach its target point. D_{max} denotes the maximum distance between any two robot positions and target points at any time. We have for δ-inner target functions, $D_{\max} \leq \text{diam}(\text{minbox}(P(0) \cup C(0))$.

Lemma 4. *In the asynchronous LCM_S model with δ-inner target functions, after any round at time interval $[t_{start}, t_{end}]$, it holds for every dimension k with $D_k(t_{start}) > 0$ that at least one of the following two intervals is empty: $\Gamma_L := [a_k(t_{start}), a_k(t_{start}) + \beta)$ or $\Gamma_U := (b_k(t_{start}) - \beta, b_k(t_{start})]$ with $\beta := \min\{1, \frac{S}{D_{max}}\} \cdot |\Delta_{L,k}(t_{start})|$.*

Proof. Set $\Delta_L := \Delta_{L,k}(t_{start})$ and $\Delta_U := \Delta_{U,k}(t_{start})$. By Lemma 2 we get that if during this round some target points are calculated in Δ_L *and* Δ_U, then one of both intervals is empty at the end. The claim follows with $\Delta_L \supseteq \Gamma_L$ and $\Delta_U \supseteq \Gamma_U$.

Now assume that in one of the intervals, w.l.o.g. assume this to be Δ_L, no target point is calculated during the whole round. So it holds for all time points $t \in [t_{start}, t_{end}]$ and for all robots $r \in R_{\text{LOOK}}(t)$ that $c_{r,k}(t) \notin \Delta_L$. Since all robots perform at least one update during this round, at time t_{end} no robot's target point is in Δ_L. Hence, at time t_{end} the interval Δ_L is either empty or it contains robot positions only but no target points.

Set $E := \{r \in R \mid p_{r,k}(t_{end}) \in \Delta_L\}$. We already know that $c_{r,k}(t_{end}) \notin \Delta_L$. Thus, all robots in E performed their last movement inside Δ_L towards an unreached target point outside Δ_L, thus, by a distance of at least S.

Next, we analyze these movements in their projection onto coordinate axis k, where we are interested in lower-bounding the minimum final position $a_k(t_{end}) = \min\{p_{i,k}(t_{end}) \mid i \in E\} = a_k(t_{start}) + \gamma$ for some $\gamma > 0$, by determining the minimal possible increasement γ. Although the movement was by a distance of at least S, in the projection onto this coordinate axis it might be $\gamma \ll S$.

However, γ cannot be arbitrary small. In fact, by geometric arguments, γ is minimal in the following scenario: Let the robot i perform the update of its last cycle in this round at time t_0 at position $p_i(t_0) := (a_1(t_{start}), \dots, a_d(t_{start}))$ and let it calculate the target point $c_i(t_0) := (z_1, \dots, z_d) \notin \Delta_L$ with:

$$z_l = \begin{cases} b_l - \delta \cdot D_{k+1}(t_{start}) & k \neq l \\ a_k(t_{start}) + |\Delta_L| & k = l \end{cases}$$

This z_l is along dimension k the lowest possible target point outside of Δ_L, while yielding the lowest possible increasement along this dimension during the next move. Here most of the contribution of its move distance S is hidden in the other $d-1$ dimensions. By this, $p_i(t_{end})$ is minimal. The Intercept Theorem gives:

$$\frac{\gamma}{|\Delta_L|} = \frac{S}{|c_i(t_0) - p_i(t_0)|} \quad \Rightarrow \quad \gamma = \frac{|\Delta_L| \cdot S}{|c_i(t_0) - p_i(t_0)|} > \frac{|\Delta_L| \cdot S}{D_{max}}$$

In case of $\gamma < |\Delta_L|$ we get for all robots $r \in R$ that $p_{r,k}(t_{end}) \geq p_{i,k}(t_{end}) \geq a_k(t_{start}) + \gamma > a_k(t_{start}) + \frac{|\Delta_L| \cdot S}{D_{max}}$ and for the target points $c_{r,k}(t_{end}) \notin \Delta_L$. Thus, Γ_L is empty at time t_{end}.

In case of $\gamma \geq |\Delta_L|$ we get for all robots $r \in R$ that $p_{r,k}(t_{end}) \geq a_k(t_{start}) + |\Delta_L|$ and again $c_{r,k}(t_{end}) \notin \Delta_L$. Thus, $\Delta_L \supseteq \Gamma_L$ is empty at time t_{end}. □

Theorem 2. *During each round in the asynchronous LCM_S model with δ-inner target functions, the diameter of the minbox around all target points and robot positions is halved in all dimensions separately after $\mathcal{O}(\frac{1}{\delta} + \frac{D_{max}}{\delta \cdot S})$ rounds.*

Proof. For any fixed round, consider the time interval $[t_{start}, t_{end}]$ that exactly limits this round. Lemma 4 gives:

$$D_k(t_{end}) \leq \begin{cases} (1 - \frac{\delta}{2}) \cdot D_k(t_{start}) & , \Delta_L \text{ or } \Delta_U \text{ is empty at time } t_{end} \\ (1 - \frac{\delta \cdot S}{2 \cdot D_{max}}) \cdot D_k(t_{start}) & , \text{otherwise} \end{cases}$$

Hence, h_k is halved after at most $\mathcal{O}(\frac{1}{\delta} + \frac{D_{max}}{\delta \cdot S})$ rounds. □

3 Runtime of the Center of Gravity Algorithm

In this section we apply our analysis to a specific δ-inner target function, namely the *Center of Gravity*. This is a straight-forward approach for solving the convergence problem by computing the average point of the robot positions as next target point: $c = \frac{1}{n} \sum_{r \in R} p_r$. Its previously best known runtime bound in asynchronous LCM_∞ is $\mathcal{O}(n^2)$, in LCM_S it is $\mathcal{O}(n^2 + n\frac{h}{S})$, where h is the initial diameter of the convex hull of the robot positions and target points [6]. Its definition implicitly claims that the robots have the ability of *multiplicity detection*, i.e., they can distinguish multiple robots that currently stay at the same position. A similar formulation is available without multiplicity detection as $c = \frac{1}{|P(t)|} \sum_{p \in P(t)} p$. In contrast to [6] our results apply for both alternatives.

For calculating this target function, robots need only very weak capabilities: They may be oblivious, indistinguishable, have no communication, and we even allow any individual affine local coordinate systems for their local views.

Lemma 5. *For n robots in $\mathbb{R}^d$ with any affine individual robots' coordinate systems, the Center of Gravity with/without multiplicity detection is $1/n$-inner.*

Proof. Since the Center of Gravity $c = \frac{1}{n}\sum_{i=1}^{n} p_i$ is globally unique, it is sufficient to show the $1/n$-innerness for any arbitrary fixed dimension k of some global coordinate system. For such a k let interval $[a_k, b_k]$ denote the minbox' expanse along dimension k and c_k the Center of Gravity's projection onto dimension k. With a robot j at position b_k, we have for the lower δ-border:

$$c_k - a_k = \frac{1}{n}\sum_{i=1}^{n}(p_{i,k} - a_k) \geq \frac{1}{n}(p_{j,k} - a_k) = \frac{1}{n}(b_k - a_k)$$

Note that the $1/n$-innerness also holds without multiplicity detection, since only one robot at the minbox' borders is needed, each. □

By having proven the $1/n$-inner property of the Center of Gravity algorithm we now may apply the former convergence and runtime results.

Theorem 3. *Consider n robots with the Center of Gravity target function and h as the diameter of the initial convex hull of the robot positions and target points.*

1. *In the LCM_∞ model,* $\mathrm{diam}(P \cup C)$ *is halved within $\mathcal{O}(n)$ rounds.*
2. *In the LCM_S model,* $\mathrm{diam}(P \cup C)$ *is halved within $\mathcal{O}(n + n \cdot \frac{h}{S})$ rounds.*

Proof. Having the $1/n$-inner property for the Center of Gravity target function from the previous lemma, we get the convergence speed from Section 2. In particular, we get a runtime of $\mathcal{O}(n)$ rounds for the LCM_∞ model from Theorem 1. For the LCM_S model with minimum movement S we get $\mathcal{O}(n + n \cdot \frac{D_{max}}{S})$ rounds (Theorem 2). Further, for the Center of Gravity it holds $D_{max} \leq h$. □

In [6] a scenario construction was presented for which Center of Gravity in asynchronous LCM_∞ requires $\Omega(n)$ rounds for halving the diameter. By that lower bound and our upper bound we close the gap to a tight worst-case bound. This lower-bound construction also applies to $\mathcal{SSYNC}$ and LCM_S.

4 The Center of Minbox Algorithm

A second target function that fulfills the δ-inner property is the *Center of Minbox*. For this, we fix coordinate system axes that are used by all robots. We also use these axes for the definition of the minboxes. For the Center of Minbox algorithm, a robot always chooses the center of $\mathrm{minbox}(P(t))$ as target point. Obviously, this target function is 1/2-inner. By our techniques we obtain:

Theorem 4. *Consider n robots with the Center of Minbox target function. Let h be the diameter of the minbox of robot positions and target points at time 0.*

1. *In the LCM_∞ model, the diameter of $P \cup C$ is halved within $\mathcal{O}(1)$ rounds.*
2. *In the LCM_S model, the diameter of $P \cup C$ is halved within $\mathcal{O}(1 + \frac{h}{S})$ rounds.*

5 Conclusion and Outlook

We introduced a novel method to analyze the convergence property and speed of target functions that fulfill the δ-inner property. By this method, we changed the perspective from the analysis of specific algorithms and robot interactions to the properties of a target function itself. This enabled us to handle a whole class of various models and combinations of robot capabilities at once, where we even allow the robots to be inhomogeneous. Besides a broader application and a simpler analysis this enabled us to improve runtime estimations and to state a tight worst-case runtime bound for the Center of Gravity algorithm of $\Theta(n)$.

Convergence of mobile robots with δ-inner target functions can even be shown in a very general setting, when exchanging the minimal movement distance S by a simple fairness assumption: A robot that wants to move a finite distance must be able to do this in a finite number of steps. Although it is not possible to give runtime bounds in this setting (minimal robot movements are not guaranteed), our technique can still be used to show that the robots converge to one point.

One can think of various other properties than the δ-inner property, which are also worthwhile to be studied by our approach. For example, let the *lower/upper replace-bounds* describe the sensitivity of the target point when a single robot is displaced by some distance $\epsilon > 0$. For the Center of Gravity it holds that *lower replace bound* = *upper replace bound* = ϵ/n, which seems to be a very special characteristic compared to other target points, and which is implicitly used for the convergence proof in [6]. Any proof for the Center of Gravity that makes use of this specific characteristic cannot simply be transferred to other target functions, or to the Center of Gravity without multiplicity detection, since its lower replace-bound is zero. An exhaustive classification of target points according to various properties might lead to deeper insights into their characteristics, and allow to intentionally choose appropriate properties for the analysis.

References

1. Agmon, N., Peleg, D.: Fault-tolerant gathering algorithms for autonomous mobile robots. In: SODA 2004: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1070–1078. Society for Industrial and Applied Mathematics, Philadelphia (2004)
2. Ando, H., Suzuki, Y., Yamashita, M.: Formation agreement problems for synchronous mobile robots with limited visibility. In: Proc. IEEE Syp. of Intelligent Control, pp. 453–460 (1995)
3. Cieliebak, M., Flocchini, P., Prencipe, G., Santoro, N.: Solving the Robots Gathering Problem. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 1182–1196. Springer, Heidelberg (2003)
4. Cieliebak, M., Flocchini, P., Prencipe, G., Santoro, N.: Solving the robots gathering problem. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 1181–1196. Springer, Heidelberg (2003)
5. Cohen, R., Peleg, D.: Robot convergence via center-of-gravity algorithms. In: Kralovic, R., Sýkora, O. (eds.) SIROCCO 2004. LNCS, vol. 3104, pp. 79–88. Springer, Heidelberg (2004)

6. Cohen, R., Peleg, D.: Convergence properties of the gravitational algorithm in asynchronous robot systems. SIAM Journal on Computing 34(6), 1516–1528 (2005)
7. Cohen, R., Peleg, D.: Local spreading algorithms for autonomous robot systems. Theoretical Computer Science 399(1-2), 71–82 (2008); Structural Information and Communication Complexity (SIROCCO 2006)
8. Cord-Landwehr, A., Degener, B., Fischer, M., Hüllmann, M., Kempkes, B., Klaas, A., Kling, P., Kurras, S., Märtens, M., Meyer auf der Heide, F., Raupach, C., Swierkot, K., Warner, D., Weddemann, C., Wonisch, D.: Collisionless gathering of robots with an extent. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jefferey, K., Králović, R., Vukolić, M., Wolf, S. (eds.) SOFSEM 2011. LNCS, vol. 6543, pp. 178–189. Springer, Heidelberg (2011)
9. Czyzowicz, J., Gąsieniec, L., Pelc, A.: Gathering few fat mobile robots in the plane. Theoretical Computer Science 410(6-7), 481–499 (2009)
10. Degener, B., Kempkes, B., Meyer auf der Heide, F.: A local $\mathcal{O}(n^2)$ gathering algorithm. In: SPAA 2010: Proceedings of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures, pp. 217–223. ACM, New York (2010)
11. Dieudonné, Y., Petit, F.: Self-stabilizing deterministic gathering. In: Dolev, S. (ed.) ALGOSENSORS 2009. LNCS, vol. 5804, pp. 230–241. Springer, Heidelberg (2009)
12. I. Suzuki, M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.
13. Izumi, T., Izumi, T., Kamei, S., Ooshita, F.: Randomized gathering of mobile robots with local-multiplicity detection. In: Guerraoui, R., Petit, F. (eds.) SSS 2009. LNCS, vol. 5873, pp. 384–398. Springer, Heidelberg (2009)
14. Izumi, T., Katayama, Y., Inuzuka, N., Wada, K.: Gathering autonomous mobile robots with dynamic compasses: An optimal result. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 298–312. Springer, Heidelberg (2007)
15. Prencipe, G.: Corda: Distributed Coordination of a Set of Autonomous Mobile Robots. PhD thesis, Universita Degli Studi Di Pisa (2002)
16. Prencipe, G.: Impossibility of gathering by a set of autonomous mobile robots. Theoretical Computer Science 384(2-3), 222–231 (2007); Structural Information and Communication Complexity (SIROCCO 2005)
17. Souissi, S., Défago, X., Yamashita, M.: Gathering asynchronous mobile robots with inaccurate compasses. In: Shvartsman, M.M.A.A. (ed.) OPODIS 2006. LNCS, vol. 4305, pp. 333–349. Springer, Heidelberg (2006)
18. Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots – formation and agreement problems. In: Proceedings of the 3rd Annual Colloquium on Structural Information and Communication Complexity (SIROCCO 1996), pp. 313–330. Carleton Scientific, Waterloo (1996)
19. Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: Formation of geometric patterns. SIAM Journal on Computing 28(4), 1347–1363 (1999)

Author Index