

ARCoSS

LNCS 9234

Giuseppe F. Italiano
Giovanni Pighizzini
Donald T. Sannella (Eds.)

Mathematical Foundations of Computer Science 2015

40th International Symposium, MFCS 2015
Milan, Italy, August 24–28, 2015
Proceedings, Part I



 Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison, UK

Josef Kittler, UK

John C. Mitchell, USA

Bernhard Steffen, Germany

Demetri Terzopoulos, USA

Gerhard Weikum, Germany

Takeo Kanade, USA

Jon M. Kleinberg, USA

Friedemann Mattern, Switzerland

Moni Naor, Israel

C. Pandu Rangan, India

Doug Tygar, USA

Advanced Research in Computing and Software Science

Subline of Lecture Notes in Computer Science

Subline Series Editors

Giorgio Ausiello, *University of Rome 'La Sapienza', Italy*

Vladimiro Sassone, *University of Southampton, UK*

Subline Advisory Board

Susanne Albers, *TU Munich, Germany*

Benjamin C. Pierce, *University of Pennsylvania, USA*

Bernhard Steffen, *University of Dortmund, Germany*

Deng Xiaotie, *City University of Hong Kong*

Jeannette M. Wing, *Microsoft Research, Redmond, WA, USA*

More information about this series at <http://www.springer.com/series/7407>

Giuseppe F. Italiano · Giovanni Pighizzini
Donald T. Sannella (Eds.)

Mathematical Foundations of Computer Science 2015

40th International Symposium, MFCS 2015
Milan, Italy, August 24–28, 2015
Proceedings, Part I

Editors

Giuseppe F. Italiano
Università di Roma "Tor Vergata"
Rome
Italy

Donald T. Sannella
University of Edinburgh
Edinburgh
UK

Giovanni Pighizzini
Università degli Studi di Milano
Milan
Italy

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-662-48056-4 ISBN 978-3-662-48057-1 (eBook)
DOI 10.1007/978-3-662-48057-1

Library of Congress Control Number: 2015945159

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

Springer Heidelberg New York Dordrecht London
© Springer-Verlag Berlin Heidelberg 2015

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made.

Printed on acid-free paper

Springer-Verlag GmbH Berlin Heidelberg is part of Springer Science+Business Media
(www.springer.com)

Preface

The series of MFCS symposia has a long and well-established tradition of encouraging high-quality research into all branches of theoretical computer science. Its broad scope provides an opportunity to bring together researchers who do not usually meet at specialized conferences. The first symposium was held in 1972. Until 2012 MFCS symposia were organized on a rotating basis in Poland, the Czech Republic, and Slovakia. The 2013 edition took place in Austria, the 2014 edition in Hungary, while in 2015 MFCS was organized for the first time in Italy.

The 40th International Symposium on Mathematical Foundations of Computer Science (MFCS 2015) was held in Milan during August 24–28, 2015. The scientific program of the symposium consisted of five invited talks and 81 contributed papers.

To celebrate the 40th edition of the conference, a special invited talk was given by:

- Zoltán Ésik (University of Szeged, Hungary)

This talk was sponsored by the European Association of Theoretical Computer Science (EATCS). The other invited talks were given by:

- Anindya Banerjee (IMDEA Software Institute, Spain)
- Paolo Boldi (University of Milan, Italy)
- Martin Kutrib (University of Giessen, Germany)
- Yishay Mansour (Microsoft Research, Hertzelia and Tel Aviv University)

We are grateful to all invited speakers for accepting our invitation and for their excellent presentations at the symposium.

The 81 contributed papers were selected by the Program Committee (PC) out of a total of 201 submissions. All submitted papers were peer reviewed and evaluated on the basis of originality, quality, significance, and presentation. To support the selection process, approximately 600 reviews were written by PC members with the help of external experts.

As is the MFCS tradition, a Best Paper Award and a Best Student Paper Award sponsored by EATCS were assigned. The PC decided to assign these awards to the following papers:

- “Strong Inapproximability of the Shortest Reset Word” by Paweł Gawrychowski and Damian Straszak (Best Paper Award)
- “Maximum Minimal Vertex Cover Parameterized by Vertex Cover” by Meirav Zehavi (Best Student Paper Award)

We thank all authors who submitted their work for consideration to MFCS 2015. We wish to thank all PC members and external reviewers for their competent and timely handling of the submissions. The success of the scientific program is due to their hard work. During the selection process and for preparing these proceedings, we used the EasyChair conference management system, which provided excellent support.

Owing to the large number of accepted papers, the proceedings of the conference were divided into two volumes on a thematic basis: *Logic, Semantics, Automata and Theory of Programming* (Vol. I) and *Algorithms, Complexity and Games* (Vol. II).

We gratefully acknowledge the support of the University of Milan (Università degli Studi di Milano, Dipartimento di Informatica) and EATCS. Special thanks for the local organization are due to Violetta Lonati (University of Milan). We also thank Bruno Guillon (University Paris-Diderot, France) for the website design and maintenance.

June 2015

Giuseppe F. Italiano
Giovanni Pighizzini
Don Sannella

Conference Organization

Program Committee Chairs

Giuseppe F. Italiano, co-chair	University of Rome “Tor Vergata”, Italy
Giovanni Pighizzini, chair	University of Milan, Italy
Donald Sannella, co-chair	University of Edinburgh, UK

Program Committee

Hee-Kap Ahn	POSTECH, Korea
Andris Ambainis	University of Latvia
Marie-Pierre Béal	University of Paris-Est Marne-la-Vallée, France
Lars Birkedal	Aarhus University, Denmark
Jarosław Byrka	University of Wrocław, Poland
Luis Caires	University of Lisbon “Nova”, Portugal
Bruno Codenotti	CNR Pisa, Italy
Adriana Compagnoni	Stevens Institute of Technology, USA
Erzsébet Csuha-j-Varjú	Eötvös Loránd University, Budapest, Hungary
Artur Czumaj	University of Warwick, UK
Rocco de Nicola	IMT Lucca, Italy
Martin Dietzfelbinger	Technical University of Ilmenau, Germany
Devdatt Dubashi	Chalmers, Sweden
Amos Fiat	Tel Aviv University, Israel
Enrico Formenti	Nice Sophia Antipolis University, France
Pierre Fraigniaud	CNRS and University Paris Diderot, France
Matt Franklin	UC Davis, USA
Loukas Georgiadis	University of Ioannina, Greece
Jan Holub	Czech Technical University in Prague, Czech Republic
Markus Holzer	University of Giessen, Germany
Martin Lange	University of Kassel, Germany
Massimo Lauria	KTH Royal Institute of Technology, Sweden
Inge Li Gørtz	Technical University of Denmark
Alberto Marchetti-Spaccamela	University of Rome “La Sapienza”, Italy
Elvira Mayordomo	University of Zaragoza, Spain

Pierre McKenzie	University of Montréal, Canada
Friedhelm Meyer auf der Heide	University of Paderborn, Germany
Prakash Panangaden	McGill University, Canada
Dana Pardubská	Comenius University, Bratislava, Slovakia
Kunsoo Park	Seoul National University, Korea
Alexander Rabinovich	Tel Aviv University, Israel
Rajeev Raman	University of Leicester, UK
Jean-Francois Raskin	University of Brussels “Libre”, Belgium
Liam Roditty	Bar-Ilan University, Israel
Marie-France Sagot	Inria and University of Lyon 1, France
Piotr Sankowski	University of Warsaw, Poland
Philippe Schnoebelen	LSV, CNRS and ENS Cachan, France
Marinella Sciortino	University of Palermo, Italy
Jiří Sgall	Charles University, Prague, Czech Republic
Arseny Shur	Ural Federal University, Russia
Mariya Soskova	Sofia University, Bulgaria
Tarmo Uustalu	Tallinn University of Technology, Estonia
Peter van Emde Boas	University of Amsterdam, The Netherlands
Jan van Leeuwen	Utrecht University, The Netherlands
Dorothea Wagner	Karlsruhe Institute of Technology, Germany
Peter Widmayer	ETH Zürich, Switzerland
Jiří Wiedermann	Academy of Sciences, Czech Republic
Christos Zaroliagis	University of Patras, Greece
Norbert Zeh	Dalhousie University, Halifax, Canada

Steering Committee

Juraj Hromkovič	ETH Zürich, Switzerland
Antonín Kučera, chair	Masaryk University, Czech Republic
Jerzy Marcinkowski	University of Wrocław, Poland
Damian Niwiński	University of Warsaw, Poland
Branislav Rovan	Comenius University, Bratislava, Slovakia
Jiří Sgall	Charles University, Prague, Czech Republic

Additional Reviewers

Akutsu, Tatsuya	Barto, Libor	Bevern, René van
Allender, Eric	Bärtschi, Andreas	Beyersdorff, Olaf
Almeida, Jorge	Basset, Nicolas	Bi, Jingguo
Amir, Amihod	Baum, Moritz	Bianchi, Maria Paola
Ananichev, Dmitry	Becchetti, Luca	Bienvenu, Laurent
Asarin, Eugene	Berkholz, Christoph	Bille, Philip
Azar, Yossi	Bernasconi, Anna	Bioglio, Livio
Bampas, Evangelos	Bernstein, Aaron	Bläsius, Thomas

Blondin, Michael	Doyen, Laurent	Hamann, Michael
Blumensath, Achim	Drees, Maximilian	Haviv, Ishay
Boella, Guido	Droste, Manfred	Hoogeboom, Hendrik Jan
Böhm, Martin	Drucker, Andrew	Hoyrup, Mathieu
Bohmova, Katerina	Đuriš, Pavol	Hrubes, Pavel
Boker, Udi	Eden, Alon	Huang, Chien-Chung
Bollig, Benedikt	Englert, Matthias	Huang, Sangxia
Bonacina, Ilario	Eppstein, David	Hundeshagen, Norbert
Bonelli, Eduardo	Epstein, Leah	Iliev, Petar
Borassi, Michele	Erde, Joshua	Itsykson, Dmitry
Bosek, Bartłomiej	Fasoulakis, Michail	Jansen, Klaus
Bozianu, Rodica	Feldotto, Matthias	Jeandel, Emmanuel
Bradfield, Julian	Fenner, Stephen	Jecker, Ismaël
Bradley, Jeremy	Fici, Gabriele	Jež, Łukasz
Bremer, Joachim	Fijalkow, Nathanaël	Johannsen, Jan
Bresolin, Davide	Filiot, Emmanuel	Johnson, Matthew
Breveglieri, Luca	Flammini, Michele	Jones, Mark
Bruse, Florian	Forejt, Vojtech	Jung, Daniel
Bulteau, Laurent	Forišek, Michal	Kari, Jarkko
Cadilhac, Michaël	Franciosa, Paolo	Kavitha, Telikepalli
Canonne, Clément	Frid, Anna	Kempa, Dominik
Carayol, Arnaud	Frigioni, Daniele	Kernberger, Daniel
Cardinal, Jean	Fuchs, Fabian	Kikot, Stanislav
Carpi, Arturo	Fukuda, Komei	Kim, Min-Gyu
Cassez, Franck	Gajardo, Anahi	Kim, Sang-Sub
Caucal, Didier	Galesi, Nicola	Kis, Tamas
Cave, Andrew	Gavinsky, Dmitry	Klasing, Ralf
Cerone, Andrea	Gazdag, Zsolt	Klein, Kim-Manuel
Čevorová, Kristína	Giannopoulos, Panos	Komusiewicz, Christian
Chailloux, André	Giannopoulou, Georgia	Kontogiannis, Spyros
Chechik, Shiri	Girard, Vincent	Kopczynski, Eryk
Cho, Dae-Hyung	Gogacz, Tomasz	Korman, Matias
Cicalese, Ferdinando	Goldenberg, Elazar	Koucký, Michal
Cleophas, Loek	Goldwurm, Massimiliano	Koutris, Paraschos
Colcombet, Thomas	Göller, Stefan	Krajíček, Jan
Cording, Patrick Hagge	Gordon, Colin S.	Královič, Rastislav
Dal Lago, Ugo	Goubault-Larrecq, Jean	Krebs, Andreas
Damaschke, Peter	Green, Fred	Kuich, Werner
D'Angelo, Gianlorenzo	Grigorieff, Serge	Kulkarni, Janardhan
Davies, Peter	Grosshans, Nathan	Kumar, Mrinal
Dell, Holger	Grossi, Giuliano	La Torre, Salvatore
Della Monica, Dario	Guillon, Pierre	Laura, Luigi
Dennunzio, Alberto	Guo, Heng	Lázár, Katalin A.
Dibbelt, Julian	Gusev, Vladimir	Lazic, Ranko
Diestel, Reinhard	Habib, Michel	Li, Shouwei
Dobrev, Stefan	Halldorsson, Magnus M.	Limouzy, Vincent

Lin, Jianyi	Papadopoulos, Charis	Smith, Adam
Löding, Christof	Parotsidis, Nikos	Son, Wanbin
Loff, Bruno	Paryen, Haim	Sornat, Krzysztof
Lombardy, Sylvain	Paul, Christophe	Spoerhase, Joachim
López-Ortiz, Alejandro	Pavlogiannis, Andreas	Srinivasan, Srikanth
Loreti, Michele	Pich, Ján	Stougie, Leen
MacKenzie, Kenneth	Pilipczuk, Marcin	Suchy, Ondřej
Mäcker, Alexander	Ponse, Alban	Taati, Siamak
Mahajan, Meena	Pouly, Amaury	Tagliaferri, Roberto
Malatyali, Manuel	Praveen, M.	Tan, Tony
Mamagishvili, Akaki	Pribavkina, Elena	Thapen, Neil
Mandrioli, Dino	Protti, Fabio	Tichler, Krisztián
Marathe, Madhav	Provillard, Julien	Tiezzi, Francesco
Markarian, Christine	Prutkin, Roman	Todinca, Ioan
Martens, Wim	Rabinovitch, Alex	Torres Vieira, Hugo
Martin, Russell	Ramanujan, M.S.	Tribastone, Mirco
Mary, Arnaud	Ramyaa, Ramyaa	Trystram, Denis
Massazza, Paolo	Randour, Mickael	Ucar, Bora
Mazoit, Frédéric	Rao, Michaël	Uznański, Przemysław
Mederly, Pavol	Rasin, Oleg	Vaananen, Jouko
Meduna, Alexander	Regan, Kenneth	van Leeuwen, Erik Jan
Mehrabi, Ali D.	Restivo, Antonio	Vandin, Andrea
Mendes de Oliveira, Rafael	Riveros, Cristian	Vanier, Pascal
Mertzios, George	Romashchenko, Andrei	Velner, Yaron
Mezzina, Claudio Antares	Rutter, Ignaz	Veselý, Pavel
Miksa, Mladen	Rybicki, Bartosz	Vinyals, Marc
Miller, Joseph S.	Sabharwal, Yogish	Vollmer, Heribert
Montanari, Angelo	Salo, Ville	Wacker, Arno
Moscardelli, Luca	Salvail, Louis	Ward, Justin
Müller, Moritz	Saurabh, Saket	Watrigant, Rémi
Mundhenk, Martin	Schaudt, Oliver	Watrous, John
Nakagawa, Kotaro	Schewe, Klaus-Dieter	Weihrauch, Klaus
Obraztsova, Svetlana	Schmid, Markus L.	Williams, Ryan
Oh, Eunjin	Schmidt, Jens M.	Wollan, Paul
Okhotin, Alexander	Schmitz, Sylvain	Yakaryilmaz, Abuzer
Otachi, Yota	Seki, Shinnosuke	Yoon, Sang-Duk
Ott, Sebastian	Serre, Olivier	Zeitoun, Marc
Otto, Martin	Seto, Kazuhisa	Ziadi, Tewfik
Oum, Sang-II	Shen, Alexander	Zielinski, Pawel
Paluch, Katarzyna	Shpilka, Amir	Živný, Stanislav
Panagiotou, Konstantinos	Siggers, Mark	Zündorf, Tobias
Pantziou, Grammati	Slaman, Theodore	
	Sloth, Christoffer	

Invited Contributions

Modular Reasoning for Behavior-Preserving Data Structure Refactorings

Anindya Banerjee

IMDEA Software Institute, Spain
anindya.banerjee@imdea.org

Abstract. A properly encapsulated data structure can be revised for refactoring without affecting the behaviors of clients of the data structure. Encapsulation ensures that clients are representation independent, that is, their behaviors are independent of particular choices of data structure representations. Modular reasoning about data structure revisions in heap-manipulating programs, however, is a challenge because encapsulation in the presence of shared mutable objects is difficult to ensure for a variety of reasons.

- Pointer aliasing can break encapsulation and invalidate data structure invariants.
- Representation independence is nontrivial to guarantee in a generic manner, without recourse to specialized disciplines such as ownership.
- Mechanical verification of representation independence using theorem provers is nontrivial because it requires relational reasoning between two different data structure representations. Such reasoning lies outside the scope of most modern verification tools.

We address the challenge by reasoning in Region Logic [1, 2], a Hoare logic augmented with state dependent “modifies” specifications based on simple notations for object sets, termed “regions”. Region Logic uses ordinary first order logic assertions to support local reasoning and also the hiding of invariants on encapsulated state, in ways suited to verification using SMT solvers. By using relational assertions, the logic can reason about behavior-preservation of data structure refactorings even in settings where full functional pre/post specifications are absent. The key ingredient behind such reasoning is a new proof rule that embodies representation independence.

This work is in collaboration with David A. Naumann and Mohammad Nikouei (Stevens Institute of Technology).

References

1. Banerjee, A., Naumann, D.A., Rosenberg, S.: Local reasoning for global invariants, part I: Region logic. *J. ACM*, **60**(3), 18:1–18: 56 (2013)
2. Banerjee, A., Naumann, D.A.: Local reasoning for global invariants, part II: Dynamic boundaries. *J. ACM*, **60**(3), 19:1–19:73 (2013)

Minimal and Monotone Minimal Perfect Hash Functions

Paolo Boldi

Dipartimento di Informatica,
Università degli Studi di Milano, Milan, Italy

Abstract. A minimal perfect hash function (MPHF) is a (data structure providing a) bijective map from a set S of n keys to the set of the first n natural numbers. In the static case (i.e., when the set S is known in advance), there is a wide spectrum of solutions available, offering different trade-offs in terms of construction time, access time and size of the data structure. MPHFs have been shown to be useful to compress data in several data management tasks. In particular, *order-preserving* minimal perfect hash functions have been used to retrieve the position of a key in a given list of keys: however, the ability to preserve any given order leads to an unavoidable $\Omega(n \log n)$ lower bound on the number of bits required to store the function. Recently, it was observed that very frequently the keys to be hashed are sorted in their intrinsic (i.e., lexicographical) order. This is typically the case of dictionaries of search engines, list of URLs of web graphs, etc. MPHFs that preserve the intrinsic order of the keys are called *monotone* (MMPHF). The problem of building MMPHFs is more recent and less studied (for example, no lower bounds are known) but once more there is a wide spectrum of solutions available, by now. In this paper, we survey some of the most practical techniques and tools for the construction of MPHFs and MMPHFs.

Equational Properties of Fixed Point Operations in Cartesian Categories: An Overview

Zoltán Ésik

Department of Computer Science, University of Szeged, Szeged, Hungary

Abstract. Several fixed point models share the equational properties of iteration theories, or iteration categories, which are cartesian categories equipped with a fixed point or dagger operation subject to certain axioms. After discussing some of the basic models, we provide equational bases for iteration categories and offer an analysis of the axioms. Although iteration categories have no finite base for their identities, there exist finitely based implicational theories that capture their equational theory. We exhibit several such systems. Then we enrich iteration categories with an additive structure and exhibit interesting cases where the interaction between the iteration category structure and the additive structure can be captured by a finite number of identities. This includes the iteration category of monotonic or continuous functions over complete lattices equipped with the least fixed point operation and the binary supremum operation as addition, the categories of simulation, bisimulation, or language equivalence classes of processes, context-free languages, and others. Finally, we exhibit a finite equational system involving residuals, which is sound and complete for monotonic or continuous functions over complete lattices in the sense that it proves all of their identities involving the operations and constants of cartesian categories, the least fixed point operation and binary supremum, but not involving residuals.

Reversible and Irreversible Computations of Deterministic Finite-State Devices

Martin Kutrib

Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
kutrib@informatik.uni-giessen.de

Abstract. Finite-state devices with a read-only input tape that may be equipped with further resources as queues or pushdown stores are considered towards their ability to perform reversible computations. Some aspects of the notion of logical reversibility are addressed. We present some selected results on the decidability, uniqueness, and size of minimal reversible deterministic finite automata. The relations and properties of reversible automata that are equipped with storages are discussed, where we exemplarily stick with the storage types queue and pushdown store. In particular, the computational capacities, decidability problems, and closure properties are the main topics covered, and we draw attention to the overall picture and some of the main ideas involved.

Robust Inference and Local Algorithms

Yishay Mansour

Microsoft Research, Hertzelia and Tel-Aviv University, Hertzelia, Israel

Abstract. We introduce a new feature to inference and learning which we call *robustness*. By robustness we intuitively model the case that the observation of the learner might be corrupted. We survey a new and novel approach to model such possible corruption as a zero-sum game between an adversary that selects the corruption and a learner that predicts the correct label. The corruption of the observations is done in a worst-case setting, by an adversary, where the main restriction is that the adversary is limited to use one of a fixed known class of modification functions. The main focus in this line of research is on *efficient* algorithms both for the inference setting and for the learning setting. In order to be efficient in the dimension of the domain, one cannot hope to inspect all the possible inputs. For this, we have to invoke local computation algorithms, that inspect only a logarithmic fraction of the domain per query.

Contents – Part I

Invited Contributions

Minimal and Monotone Minimal Perfect Hash Functions	3
<i>Paolo Boldi</i>	
Equational Properties of Fixed Point Operations in Cartesian Categories: An Overview	18
<i>Zoltán Ésik</i>	
Reversible and Irreversible Computations of Deterministic Finite-State Devices	38
<i>Martin Kutrib</i>	
Robust Inference and Local Algorithms	53
<i>Yishay Mansour</i>	

Logic, Semantics, Automata and Theory of Programming

Uniform Generation in Trace Monoids.	63
<i>Samy Abbes and Jean Mairesse</i>	
When Are Prime Formulae Characteristic?	76
<i>L. Aceto, D. Della Monica, I. Fábregas, and A. Ingólfssdóttir</i>	
Stochastization of Weighted Automata	89
<i>Guy Avni and Orna Kupferman</i>	
Algebraic Synchronization Criterion and Computing Reset Words.	103
<i>Mikhail Berlinkov and Marek Szykula</i>	
Recurrence Function on Sturmian Words: A Probabilistic Study	116
<i>Valérie Berthé, Eda Cesaratto, Pablo Rotondo, Brigitte Vallée, and Alfredo Viola</i>	
Exponential-Size Model Property for PDL with Separating Parallel Composition.	129
<i>Joseph Boudou</i>	
A Circuit Complexity Approach to Transductions	141
<i>Michaël Cadilhac, Andreas Krebs, Michael Ludwig, and Charles Paperman</i>	

Locally Chain-Parsable Languages	154
<i>Stefano Crespi Reghizzi, Violetta Lonati, Dino Mandrioli, and Matteo Pradella</i>	
Classes of Languages Generated by the Kleene Star of a Word.	167
<i>Laure Daviaud and Charles Paperman</i>	
Relating Paths in Transition Systems: The Fall of the Modal Mu-Calculus . . .	179
<i>Cătălin Dima, Bastien Maubert, and Sophie Pinchinat</i>	
Weighted Automata and Logics on Graphs.	192
<i>Manfred Droste and Stefan Dück</i>	
Longest Gapped Repeats and Palindromes	205
<i>Marius Dumitran and Florin Manea</i>	
Quasiperiodicity and Non-computability in Tilings	218
<i>Bruno Durand and Andrei Romashchenko</i>	
The Transitivity Problem of Turing Machines.	231
<i>Anahí Gajardo, Nicolas Ollinger, and Rodrigo Torres-Avilés</i>	
Strong Inapproximability of the Shortest Reset Word.	243
<i>Paweł Gawrychowski and Damian Straszak</i>	
Finitary Semantics of Linear Logic and Higher-Order Model-Checking	256
<i>Charles Grellois and Paul-André Melliès</i>	
Complexity of Propositional Independence and Inclusion Logic	269
<i>Miika Hannula, Juha Kontinen, Jonni Virtema, and Heribert Vollmer</i>	
Modal Inclusion Logic: Being Lax is Simpler than Being Strict	281
<i>Lauri Hella, Antti Kuusisto, Arne Meier, and Heribert Vollmer</i>	
Differential Bisimulation for a Markovian Process Algebra.	293
<i>Giulio Iacobelli, Mirco Tribastone, and Andrea Vandin</i>	
On the Hardness of Almost-Sure Termination	307
<i>Benjamin Lucien Kaminski and Joost-Pieter Katoen</i>	
Graphs Identified by Logics with Counting	319
<i>Sandra Kiefer, Pascal Schweitzer, and Erkal Selman</i>	
Synchronizing Automata with Extremal Properties	331
<i>Andrzej Kisielewicz and Marek Szykuła</i>	
Ratio and Weight Quantiles	344
<i>Daniel Krähmann, Jana Schubert, Christel Baier, and Clemens Dubsclaff</i>	

Precise Upper and Lower Bounds for the Monotone Constraint Satisfaction Problem	357
<i>Victor Lagerkvist</i>	
Definability by Weakly Deterministic Regular Expressions with Counters is Decidable	369
<i>Markus Latte and Matthias Niewerth</i>	
On the Complexity of Reconfiguration in Systems with Legacy Components	382
<i>Jacopo Mauro and Gianluigi Zavattaro</i>	
Eliminating Recursion from Monadic Datalog Programs on Trees	394
<i>Filip Mazowiecki, Joanna Ochremiak, and Adam Witkowski</i>	
Computability on the Countable Ordinals and the Hausdorff-Kuratowski Theorem (Extended Abstract)	407
<i>Arno Pauly</i>	
Emergence on Decreasing Sandpile Models	419
<i>Kévin Perrot and Éric Rémila</i>	
Lost in Self-Stabilization	432
<i>Damien Regnault and Éric Rémila</i>	
Equations and Coequations for Weighted Automata.	444
<i>Julian Salamanca, Marcello Bonsangue, and Jan Rutten</i>	
Author Index	457

Contents – Part II

Near-Optimal Asymmetric Binary Matrix Partitions	1
<i>Fidaa Abed, Ioannis Caragiannis, and Alexandros A. Voudouris</i>	
Dual VP Classes	14
<i>Eric Allender, Anna Gál, and Ian Mertz</i>	
On Tinhofer’s Linear Programming Approach to Isomorphism Testing	26
<i>V. Arvind, Johannes Köbler, Gaurav Rattan, and Oleg Verbitsky</i>	
On the Complexity of Noncommutative Polynomial Factorization	38
<i>V. Arvind, Gaurav Rattan, and Pushkar Joglekar</i>	
An Algebraic Proof of the Real Number PCP Theorem	50
<i>Martijn Baartse and Klaus Meer</i>	
On the Complexity of Hub Labeling (Extended Abstract).	62
<i>Maxim Babenko, Andrew V. Goldberg, Haim Kaplan, Ruslan Savchenko, and Mathias Weller</i>	
On the Complexity of Speed Scaling	75
<i>Neal Barcelo, Peter Kling, Michael Nugent, Kirk Pruhs, and Michele Scquizzato</i>	
Almost All Functions Require Exponential Energy	90
<i>Neal Barcelo, Michael Nugent, Kirk Pruhs, and Michele Scquizzato</i>	
On Dynamic DFS Tree in Directed Graphs	102
<i>Surender Baswana and Keerti Choudhary</i>	
Metric Dimension of Bounded Width Graphs	115
<i>Rémy Belmonte, Fedor V. Fomin, Petr A. Golovach, and M.S. Ramanujan</i>	
Equality, Revisited	127
<i>Ralph Bottesch, Dmitry Gavinsky, and Hartmut Klauck</i>	
Bounding the Clique-Width of H -free Chordal Graphs.	139
<i>Andreas Brandstädt, Konrad K. Dabrowski, Shenwei Huang, and Daniël Paulusma</i>	

New Bounds for the CLIQUE-GAP Problem Using Graph Decomposition Theory	151
<i>Vladimir Braverman, Zaoxing Liu, Tejasvam Singh, N.V. Vinodchandran, and Lin F. Yang</i>	
QMA with Subset State Witnesses	163
<i>Alex Bredariol Grilo, Iordanis Kerenidis, and Jamie Sikora</i>	
Phase Transition for Local Search on Planted SAT	175
<i>Andrei A. Bulatov and Evgeny S. Skvortsov</i>	
Optimal Bounds for Estimating Entropy with PMF Queries	187
<i>Cafer Caferov, Barış Kaya, Ryan O’Donnell, and A.C. Cem Say</i>	
Mutual Dimension and Random Sequences	199
<i>Adam Case and Jack H. Lutz</i>	
Optimal Algorithms and a PTAS for Cost-Aware Scheduling	211
<i>Lin Chen, Nicole Megow, Roman Rischke, Leen Stougie, and José Verschae</i>	
Satisfiability Algorithms and Lower Bounds for Boolean Formulas over Finite Bases	223
<i>Ruiwen Chen</i>	
Randomized Polynomial Time Protocol for Combinatorial Slepian-Wolf Problem	235
<i>Daniyar Chumbalov and Andrei Romashchenko</i>	
Network Creation Games: Think Global – Act Local	248
<i>Andreas Cord-Landwehr and Pascal Lenzner</i>	
Oblivious Transfer from Weakly Random Self-Reducible Public-Key Cryptosystem	261
<i>Claude Crépeau and Raza Ali Kazmi</i>	
Efficient Computations over Encrypted Data Blocks	274
<i>Giovanni Di Crescenzo, Brian Coan, and Jonathan Kirsch</i>	
Polynomial Kernels for Weighted Problems	287
<i>Michael Etscheid, Stefan Kratsch, Matthias Mnich, and Heiko Röglin</i>	
A Shortcut to (Sun)Flowers: Kernels in Logarithmic Space or Linear Time . . .	299
<i>Stefan Fafianie and Stefan Kratsch</i>	
Metastability of Asymptotically Well-Behaved Potential Games: (Extended Abstract)	311
<i>Diodato Ferraioli and Carmine Ventre</i>	

The Shifted Partial Derivative Complexity of Elementary Symmetric Polynomials	324
<i>Hervé Fournier, Nutan Limaye, Meena Mahajan, and Srikanth Srinivasan</i>	
Parameterized Algorithms for Parity Games	336
<i>Jakub Gajarský, Michael Lampis, Kazuhisa Makino, Valia Mitsou, and Sebastian Ordyniak</i>	
Algorithmic Applications of Tree-Cut Width	348
<i>Robert Ganian, Eun Jung Kim, and Stefan Szeider</i>	
Log-Concavity and Lower Bounds for Arithmetic Circuits	361
<i>Ignacio García-Marco, Pascal Koiran, and Sébastien Tavenas</i>	
Easy Multiple-Precision Divisors and Word-RAM Constants	372
<i>Torben Hagerup</i>	
Visibly Counter Languages and the Structure of NC^1	384
<i>Michael Hahn, Andreas Krebs, Klaus-Jörn Lange, and Michael Ludwig</i>	
The Price of Connectivity for Cycle Transversals	395
<i>Tatiana R. Hartinger, Matthew Johnson, Martin Milanič, and Daniël Paulusma</i>	
Upper and Lower Bounds on Long Dual Paths in Line Arrangements	407
<i>Udo Hoffmann, Linda Kleist, and Tillmann Miltzow</i>	
A Numbers-on-Foreheads Game	420
<i>Sune K. Jakobsen</i>	
Faster Lightweight Lempel-Ziv Parsing	432
<i>Dmitry Kosolobov</i>	
Parallel Identity Testing for Skew Circuits with Big Powers and Applications	445
<i>Daniel König and Markus Lohrey</i>	
On Probabilistic Space-Bounded Machines with Multiple Access to Random Tape	459
<i>Debasis Mandal, A. Pavan, and N.V. Vinodchandran</i>	
Densest Subgraph in Dynamic Graph Streams	472
<i>Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu</i>	
The Offline Carpool Problem Revisited	483
<i>Saad Mneimneh and Saman Farhat</i>	

On Sampling Simple Paths in Planar Graphs According to Their Lengths	493
<i>Sandro Montanari and Paolo Penna</i>	
Degree-Constrained Subgraph Reconfiguration is in P	505
<i>Moritz Mühlenthaler</i>	
Generalized Pseudoforest Deletion: Algorithms and Uniform Kernel	517
<i>Geevarghese Philip, Ashutosh Rai, and Saket Saurabh</i>	
Efficient Equilibria in Polymatrix Coordination Games	529
<i>Mona Rahn and Guido Schäfer</i>	
Finding Consensus Strings with Small Length Difference Between Input and Solution Strings	542
<i>Markus L. Schmid</i>	
Active Linking Attacks	555
<i>Henning Schnoor and Oliver Woizekowski</i>	
On the Complexity of Master Problems	567
<i>Martijn van Ee and René Sitters</i>	
Efficient Algorithm for Computing All Low s - t Edge Connectivities in Directed Graphs	577
<i>Xiaowei Wu and Chenzi Zhang</i>	
Maximum Minimal Vertex Cover Parameterized by Vertex Cover.	589
<i>Meirav Zehavi</i>	
Fast Dynamic Weight Matchings in Convex Bipartite Graphs.	601
<i>Quan Zu, Miaomiao Zhang, and Bin Yu</i>	
Author Index	613

Invited Contributions

Minimal and Monotone Minimal Perfect Hash Functions

Paolo Boldi^(✉)

Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy
paolo.boldi@unimi.it

Abstract. A minimal perfect hash function (MPHF) is a (data structure providing a) bijective map from a set S of n keys to the set of the first n natural numbers. In the static case (i.e., when the set S is known in advance), there is a wide spectrum of solutions available, offering different trade-offs in terms of construction time, access time and size of the data structure. MPHFs have been shown to be useful to compress data in several data management tasks. In particular, *order-preserving* minimal perfect hash functions have been used to retrieve the position of a key in a given list of keys: however, the ability to preserve any given order leads to an unavoidable $\Omega(n \log n)$ lower bound on the number of bits required to store the function. Recently, it was observed that very frequently the keys to be hashed are sorted in their intrinsic (i.e., lexicographical) order. This is typically the case of dictionaries of search engines, list of URLs of web graphs, etc. MPHFs that preserve the intrinsic order of the keys are called *monotone* (MMPHF). The problem of building MMPHFs is more recent and less studied (for example, no lower bounds are known) but once more there is a wide spectrum of solutions available, by now. In this paper, we survey some of the most practical techniques and tools for the construction of MPHFs and MMPHFs.

1 Introduction

A minimal perfect hash function maps bijectively a set S of n keys into the set $\{0, 1, \dots, n-1\}$ of the first n natural numbers. The construction of such functions in the static case (i.e., when S is known in advance and does not change) was widely studied in the last years, leading to fundamental theoretical results such as [1, 2]. When comparing different techniques for building MPHFs one should be aware of the trade-offs between construction time, evaluation time and space needed to store the function. A general tenet is that evaluation should happen in constant time (with respect to n), whereas construction is only required to be feasible in a practical sense. The amount of space occupied by the data structure is often the most important aspect when a construction is taken into consideration; usually space is computed in an exact (i.e., non-asymptotic) way. Some exact space lower bounds for this problem are known (they are pure space bounds and do not consider evaluation time). One fundamental question is how close to the space lower bound $n \log e + \log \log u$ one can stay if the evaluation must be performed in constant time.

In this paper, I will describe two practical solutions: the first one provides a structure that is simple, constant-time, asymptotically space-optimal (i.e., $O(n)$); its actual space requirement is about twice the lower bound. The second one can potentially get closer to the lower bound, even if in practice this would require an unfeasibly long construction time; nonetheless, it provides the smallest known data structure (although it takes a long time to be built)—it occupies about 1.44 times the lower bound.

From an application-oriented viewpoint, *order-preserving* minimal perfect hash functions (OPMPHF) have been used to retrieve the position of a key in a given list of keys [3, 4]. In [5] the authors note that all existing techniques for this task assume that keys can be provided in any order, incurring an unavoidable $\Omega(n \log n)$ lower bound on the number of bits required to store the function. However, very frequently the keys to be hashed are sorted in their intrinsic (i.e., lexicographical) order. This is typically the case of dictionaries of search engines, list of URLs of web graphs, etc. [6–8]. Thus, it is interesting to study *monotone minimal perfect hash functions* (MMPHF), that map each key of a lexicographically sorted set to its ordinal position.

This problem is much younger (it was first defined and studied in [5]) and much less is known about it; in particular, no non-trivial lower bound is available as of now. I will here limit myself in describing two solutions with a different space/time trade-off.

Overall, the purpose of this brief survey is to introduce the main tools and techniques without any pretense of completeness but with the aim of showing some key points that highlight the generality of some approaches and suggest how they can be potentially used in other contexts and for different problems.

2 Definitions and Notation

Sets, Integers, Keys. For every natural number n , I let $[n] = \{0, 1, \dots, n - 1\}$; I occasionally use the same notation when n is a real number, omitting a ceiling operation.

In the following, I will always assume that a universe U of $u = |U|$ items called *keys* is fixed; this set may in many applications be infinite, but unless otherwise specified I will suppose that it is finite. Occasionally, I assume that U is endowed with a total order \leq .

In most cases, the actual nature of the keys will be irrelevant, but later on I will need to assume that u is a power of two, $U = [u]$ and that the elements of U are represented as binary strings of length $\log u$ (each $x < u$ is represented by its zero-left-padded binary expansion); in this case, \leq is the lexicographic order of binary strings (or, equivalently, the natural ordering of natural numbers). The case of non-binary strings all of the same length follows as an easy generalization; moreover, with a few modifications the stated results can apply also in terms of the *average* string length of a set of variable-length strings [9].

Hash Functions. Given a positive integer m , an m -bucket hash function for U is any function $h : U \rightarrow [m]$. The term “hash function” is often used also to refer to a data structure that computes the hash function itself.

Hash functions are an ingredient of many data structures and algorithms [10]; depending on the application, further properties may be required or desirable; often such properties are stated with respect to a specific set of keys. Given $S \subseteq U$ with $|S| = n$ and a hash function $h : U \rightarrow [m]$, one says that:

- h is *perfect* (PHF) iff it is injective;
- h is *minimal perfect* (MPHF) iff it is perfect and $n = m$;
- h is *order-preserving* with respect to some total order \preceq on U iff $x \preceq y$ implies $h(x) \leq h(y)$ for all $x, y \in S$;
- h is *monotone* iff it is order preserving with respect to \leq .

The distinction between order-preserving and monotone hash functions is not moot; there are $n!$ possible total orders on S , hence there is an obvious space lower bound of $\Omega(n \log n)$ for order-preserving hash functions. This lower bound does not hold for monotone hash functions. Notice, though, that since monotone hash functions are a special case of order-preserving hash functions (applied to the natural order), any structure for the latter can be used to implement the former, but not vice versa.

3 Generalities

Space/Time Complexity. Every hash function (data structure) requires some time and space to be built; moreover, it needs some time to be evaluated and occupies some amount of space. In this paper I consider the problem from a static viewpoint, that is, I do not care much about construction complexity (albeit I insist that construction should be feasible), but I study in detail the access space/time trade-off. While time evaluation is only determined asymptotically, I will try to make a precise (i.e., non-asymptotic) space estimate: this is essential, because in many cases all non-trivial solutions are asymptotically equivalent, and only an exact computation of the constants involved allows for a choice between alternative approaches.

Hash Randomness. In the following, I often assume that one can “draw” a hash function $h : U \rightarrow [m]$ at random; this is sometimes called “true randomness assumption”, and turns out to simplify many proofs, but it is far from realistic. The first problem is that there are m^u such hash functions, so at least $\Omega(u \log m)$ bits are needed to describe one of them; moreover, even disregarding the space occupied, the time needed for the evaluation is crucial and we need an efficient implementation of the family of hash functions we are going to use. A large stream of research is devoted to the determination of weaker conditions on the hash functions available that still allow them to be used. Although I am well aware of the problem, I am hereafter ignoring it altogether (the interested reader can find more information, e.g., in [11]). Hence, I assume that a fully random

hash function can be obtained, can be stored in constant space and evaluated in time $O((\log u)/w)$, where w is the machine-word length: in fact, $(\log u)/w$ steps are needed just to *read* a full key! Of course, if one assumes that $w = \Omega(\log u)$, hash values can be computed in constant time. The latter case will be referred to as *short-keys scenario*, whereas the general case will be called *long-keys scenario*.

Rank and Select. I will make extensive use of the two basic building blocks of several succinct data structures—rank and select. Given a bit array (or bit string) $\mathbf{b} \in \{0, 1\}^n$ (whose positions are indexed from 0) with t ones, $\text{rank}_{\mathbf{b}}(p)$ is the number of ones up to position p , exclusive ($0 \leq p \leq n$), whereas $\text{select}_{\mathbf{b}}(r)$ is the position of the r -th one in \mathbf{b} ($0 \leq r < t$). Jacobson [12] offers a constant-time implementation for the rank data structure that uses $o(n)$ additional bits besides the array \mathbf{b} ; furthermore, constant-time solutions exist that require as little space as $O(t/(\log t)^c)$ (for any desired c) over the information-theoretical lower bound $\log \binom{n}{t}$ [13]. More practical solutions, like the Elias-Fano scheme, are described in [14, 15]. For very sparse sets, Elias-Fano can be rewarding in terms of space, but the query time becomes $O(\log(n/t))$.

Signatures. The fact that a hash function $h : U \rightarrow [m]$ is an MPHf for $S \subseteq U$ does not mean that it can be used to establish membership to S ; in other words, MPHfs cannot be used as static dictionaries. If you apply h to a key *outside of* S you still get a result (a value in $[m]$) and you have no way to determine if the key was an element of S in the first place. Whether this is an issue or not really depends on your intended application. If you want to solve this problem you can, of course, combine the MPHf with a (possibly approximate) dictionary (e.g., a Bloom filter [16]) that is used to test membership to S first. An alternative, quite efficient, solution is to store an array of *signatures*: to do this, you first choose an s -bit digesting function $\sigma : U \rightarrow [2^s]$ that computes some digest of each key, and then use an array $S[-]$ of length m that, for every $x \in S$, stores $\sigma(x)$ in $S[h(x)]$. Now, if the data structure is queried with the key x , first $h(x)$ is computed and then $S[h(x)]$ is checked against $\sigma(x)$: if they coincide, $h(x)$ is returned, otherwise \perp is returned (to mean that $x \notin S$). False negatives (elements of S for which the data structure erroneously returns \perp) cannot take place, whereas the probability of false positives depends on the quality of the signature function $\sigma(-)$ and, of course, on the size s of the signature being used. At an extreme, one can sign each key with the key itself (an “identity signature”), using $s = \log u$, which makes false positives impossible.

PHF Compression. Every hash function $h : U \rightarrow [m]$ that is a PHF for the set $S \subseteq U$ can be turned into a MPHf by using an extra array of m bits \mathbf{b} , where $b_i = 1$ iff i is in the image of S through h , and by constructing an extra rank data structure on \mathbf{b} : the new MPHf $h' : U \rightarrow [n]$ is defined as $x \mapsto \text{rank}_{\mathbf{b}}(h(x))$. Note that h' has the same evaluation time as h (because the rank is computed in constant time), and uses an extra space of $m + o(m)$ bits. Moreover, if for some reason the PHF h already contains (implicitly or explicitly) a representation of

the bit array \mathbf{b} , the extra space reduces to $o(m)$ bits. This technique will be referred to as *PHF compression*.

4 Minimal Perfect Hash Functions

Fredman and Komlós proved [2] that no MPHf can occupy less than $n \log e + \log \log u + O(\log n)$ bits, as long as $n^{2+\epsilon} \leq u$; this bound is essentially tight [17, Sect.III.2.3, Thm.8], disregarding evaluation time. Approximately, if we ignore lower-order terms, this result means that we cannot use less than $\log e \approx 1.44$ bits/key.

One fundamental question is how close to the space lower bound $n \log e + \log \log u$ one can remain if the evaluation must be performed in constant time. The best theoretical results in this direction are given in [18], where a $n \log e + \log \log u + O(n(\log \log n)^2 / \log n + \log \log \log u)$ technique is provided (optimal up to an additive factor) with expected linear-time construction. The technique is only of theoretical relevance, though, as it yields a low number of bits per key only for unrealistically large values of n .

In this section, I will present two methods that are both asymptotically optimal and in practice obtain a space bound quite close to the limit. The first one, introduced in [19], can be seen as an adaptation of [4]. The second one was presented in [20] and uses a completely different approach. All the techniques presented in this section work without any change for keys of variable length. In this section, I will assume the short-keys scenario (so, hash-function evaluation takes constant time); all results easily carry over to the long-keys scenario, but evaluation of a hash function on x will require $O(|x|/w)$ steps instead.

4.1 The MWHC Construction

The authors of [4] propose a clever and simple method to build OPMPHF. In fact, although the authors do not discuss this fact, the very same method has a much more general applicability: it allows one to store an *arbitrary function* that associates an object (represented, w.l.o.g. as an r -bit vector) to each key from a fixed set. More precisely, let r be a positive integer, $S \subseteq U$ (with $n = |S|$) and $f : S \rightarrow [2^r]$ be fixed from now on. We want to build a data structure that, given any input $v \in U$ is able to output $f(v)$ when $v \in S$; the behavior of the data structure when $v \notin S$ is irrelevant. Of course, this construction can be used in particular to store any prescribed minimal perfect hash function letting $r = \log n$ and f be a bijection.

Preliminaries on Hypergraphs. Recall that a t -hypergraph $H = (V, E)$ is defined by a set of vertices V and by a set $E \subseteq \binom{V}{t}$ of¹ hyperedges. A hypergraph $H = (V, E)$ is *peelable* [4, 21] iff there exists a sequence $(e_1, x_1), \dots, (e_m, x_m)$ where

¹ I write $\binom{X}{t}$ for the set of subsets of X of cardinality t .

- e_1, \dots, e_m are all the hyperedges of H (and each hyperedge appears exactly once)
- $x_i \in e_i$ for all $i = 1, \dots, m$
- $x_i \notin e_1 \cup \dots \cup e_{i-1}$.

The sequence e_1, \dots, e_m is called the *peeling order*, and x_i is called the *hinge* of e_i .

The 2-core is the maximal subset of vertices that induce a sub-hypergraph² where all vertices have degree³ 2 or more.

Theorem 1. *A t -hypergraph is peelable iff it does not contain a 2-core. In particular, a 2-hypergraph (i.e., a graph) is peelable iff it is acyclic.*

The Construction. Let now k and m be two positive integers, to be chosen later, with $m \geq n$ (in fact, I shall choose $m = \lceil c_k \cdot n \rceil$ for some suitable $c_k \geq 1$).

Draw at random k hash functions $h_1, \dots, h_k : U \rightarrow [m]$ and consider the hypergraph $H = ([m], E)$ where $E = \{\{h_1(v), \dots, h_k(v)\} \mid v \in S\}$. Check that the following conditions hold:

- for every $v \in S$, the values $h_1(v), \dots, h_k(v)$ are all distinct (hence, H is a k -hypergraph);
- $|E| = n$ (i.e., the hyperedges corresponding to different elements of S are different);
- H is peelable.

If any of these conditions fails to hold, repeat the process and generate k new hash functions.

Theorem 2. *Under the assumptions above, consider the following system of n equations (whose indeterminates are a_0, \dots, a_{m-1}):*

$$a_{h_1(v)} + \dots + a_{h_k(v)} = f(v) \quad \forall v \in S.$$

This system has a solution $(a_0^, \dots, a_{m-1}^*)$ over \mathbf{Z}_{2^r} such that $a_x^* = 0$ if x is not a hinge.*

Proof. Consider the peelable hypergraph H defined above, and let e_1, \dots, e_m be the peeling order, and x_i be the hinge of e_i . Moreover, let w_i be the element of S corresponding to the hyperedge e_i . Consider the equations

$$a_{h_1(w_i)} + \dots + a_{h_k(w_i)} = f(w_i)$$

in order of increasing i . Each equation contains at least one variable that never appeared before (a_{x_i}): assign it so that the equation holds in \mathbf{Z}_{2^r} ; if any other previously unassigned variable appears in the equation, let it be equal to 0.

² The sub-hypergraph induced by $X \subseteq V$ is (X, E_X) where $E_X = E \cap \binom{X}{t}$.

³ The degree of a vertex is the number of hyperedges including it.

Theorem 2 implies that the computation of $f(v)$ can be easily performed (in constant time) by computing $h_1(v), \dots, h_k(v)$ and then accessing to the corresponding elements of an array that stores the solution of the system.

Choice of m and k . The choice of m determines how easy (or difficult) it will be to find a hypergraph satisfying the assumptions (in particular, peelability): if m is too small it will be hard (impossible, if $m < n$) to have a peelable graph, whereas for large values of m almost all graphs will be peelable. In fact, this is a zero-one law, i.e., there is a threshold value c_k such that if $m < c_k \cdot n$ almost no hypergraph is peelable, whereas if $m > c_k \cdot n$ almost every hypergraph is peelable. The values of c_k have been determined in [21] (following the lines of [4]) and the first few ones are $c_2 = 2$, $c_3 \approx 1.23$, $c_4 \approx 1.29$, $c_5 \approx 1.41$. It turns out, in fact, that c_3 (henceforth called $\gamma \approx 1.23$) is the minimum of the sequence, which makes $k = 3$ and $m = \lceil \gamma \cdot n \rceil$ the best choice.

Time and Space Complexity. The statement of Theorem 2 suggests that one can either store, the actual solution $(a_0^*, \dots, a_{m-1}^*)$ (that requires $mr = \gamma nr \approx 1.23 \cdot nr$ bits, i.e., $1.23r$ bits/key), or its compressed version⁴ requiring $(r + \gamma)n + o(n)$ bits (i.e., $r + 1.23$ bits/key), which is advantageous (ignoring the extra-space for the rank/select structure) when $r + \gamma < r\gamma$, i.e., when $r > 5$.

In particular, using the MWHC construction to store a minimal perfect hash ($r = \log n$) allows one to solve the OPMPHF problem in $O(n \log n)$ bits, and this is asymptotically optimal: in fact, there are $n!$ possible MPHFs, hence they require space $\Omega(n \log n)$ to be stored.

Using the MWHC Construction to Build a MPHf. [19] (in fact, independently of [4]) suggests that the same technique can be adapted to store a MPHf $h : U \rightarrow [n]$ for a set $S \subseteq U$ of n keys. Proceed as in the standard construction, but decide the function f only after finding three hash functions h_1, h_2, h_3 that produce a peelable hypergraph. Now let $r = 2$ and $f : S \rightarrow [4]$ be defined so that $h_{f(v)+1}(v)$ is the hinge of the hyperedge associated to v . By the very definition of hinge, $h_{f(v)+1}(v)$ is distinct for distinct $v \in S$, so it represents a PHF. Also observe that the proof Theorem 2 can be extended to solve the system modulo $2^r - 1$ (in our case, modulo 3); in that case, the solution $(a_0^*, \dots, a_{m-1}^*)$ (which is zero on all non-hinges) can be taken so that it is non-zero on all hinges (just use 3 instead of 0 for hinge variables, which is ok because $3 = 0$ modulo 3). Thus the m -bit vector \mathbf{b} implicitly defined as $b_i = 0$ iff $a_i^* = 0$ can be used to compress the PHF $v \mapsto h_{f(v)+1}(v)$. As a result, we obtain a MPHf using $2\gamma n + o(n)$ bits (that is, about 2.46 bits/key). This is approximately one bit larger than the lower bound (in practice, the rank data structure can be implemented so that the real number of bits is ≈ 2.65 [21]).

⁴ Keeping only the (at most) n non-zero a_i^* and storing their indices in an array on which a rank/select structure is provided.

4.2 Hash, Displace and Compress

This alternative approach, described in [20], strongly relies on the availability of a sequence $\phi_0, \phi_1, \phi_2, \dots : U \rightarrow [m]$ of independent, fully random hash functions. Although this assumption is apparently very unrealistic, there are many ways to bypass it, for example the so-called “split-and-share trick” described in [22]; in practice, even more naive approaches can do the job, as explained in the experimental section of [20].

The technique itself builds a PHF $h : U \rightarrow [(1 + \epsilon)n]$, whence a MPHf can be obtained by compression⁵. To build h a first-level hash function $g : U \rightarrow [r]$ is chosen, that divides the universe into r buckets; g also implicitly divides the set S into buckets

$$B_i = S \cap g^{-1}(i),$$

one for each $i \in [r]$. Now, for all indices i , we will determine a second-level hash function $f_i : U \rightarrow [(1 + \epsilon)n]$ picking from the sequence ϕ_0, ϕ_1, \dots . Let us say that $f_i = \phi_{\sigma(i)}$. Then h will be defined by

$$x \mapsto \phi_{\sigma(g(x))}(x).$$

Overall, to compute h one just needs to have the values of σ .

The computation of σ is done through the following steps:

- The r buckets B_i are arranged by decreasing size;
- For every i , we let $\sigma(i)$ be the first index ℓ such that ϕ_ℓ is injective on $B_0 \cup \dots \cup B_i$.

The first question is how long it takes to find an index ℓ that satisfies the last condition. With a careful analysis, it turns out that, with high probability, $\sigma(i) < C \log n$ for some suitable constant C , implying that the construction requires $O(n \log n)$ steps. Moreover, the very same argument means that every value of σ can be stored in $\log \log n + O(1)$ bits, so the overall space required to store the hash function uses $\log \log n$ bits per key with very high probability.

In [20], the authors move one step further, and show that the $\sigma(i)$'s have a geometrical distribution with $O(1)$ expectation. Thus, using for example the sophisticated compression schemes introduced in [23], one can compress σ in $O(n)$ bits (in expectation), thus using $O(1)$ bits per key.

Theorem 3. ([20]). The construction described above requires expected time $O(n \cdot (2^{n/r} + 1/\epsilon)^{n/r})$ and space $O((1/\epsilon)n)$.

Experimentally, suitable choices of r yield ≈ 2.05 bits/key (smaller than the ≈ 2.65 of the MWHC-based construction, but still much larger than the lower bound ≈ 1.44), although the construction is about twenty times slower than the MWHC-based one.

⁵ In [20] a variant is also discussed that directly produces a MPHf, but its construction time is no longer linear in expectation.

5 Monotone Minimal Perfect Hash Functions

From an application-oriented viewpoint, *order-preserving* minimal perfect hash functions (often dubbed OPMPHF, and pronounced “oomph”) have been used to retrieve the position of a key in a given list of keys [3,4]. Since all existing techniques for this task assume that keys can be provided in any order, there is an unavoidable $\Omega(n \log n)$ lower bound on the number of bits required to store the function. However, very frequently the keys to be hashed are sorted in their intrinsic (i.e., lexicographical) order. This is typically the case of dictionaries of search engines, list of URLs of web graphs, etc. [6–8]. Thus, it is interesting to explore *monotone minimal perfect hashing*—the problem of mapping each key of a lexicographically sorted set to its ordinal position.

The first paper addressing this problem was [5], where the authors provided two solutions with different space/time trade-off. The first solution (based on longest common prefixes) provides⁶ $O((\log u)/w)$ access but requires $O(\log \log u)$ bits/key. The second solution (based on a so-called z-fast trie) requires just $O(\log \log \log u)$ bits/key, but access time becomes $O((\log u)/w + \log \log u)$.

In this paper, I will limit myself to providing a full description of the former solution, and sketch the main ideas behind the second one. The techniques described in this section require that the keys have all the same length, but they can be adapted to keys of different length provided that S is prefix-free.

Both constructions (and also many of those described in [9]) rely on the idea of (lexicographic) *bucketing*. The set of keys S is partitioned into m buckets S_0, \dots, S_{m-1} preserving the lexicographic order; in other words, every key in the i -th bucket S_i are less than all the keys in bucket S_{i+1} .

- A special function $f : U \rightarrow [m]$ called *distributor* maps every $x \in S$ to the index of the appropriate bucket (i.e., the bucket where x belongs).
- Then, for every i , a MMPHF $h_i : U \rightarrow [|S_i|]$ is stored for the set S_i .
- Finally, a function $\ell : [m] \rightarrow [n]$ is provided such that $\ell(i) = \sum_{j < i} |S_j|$.

Clearly the function $h : U \rightarrow [n]$ defined by

$$h(x) = \ell(f(x)) + h_{f(x)}(x)$$

is a MMPHF for S . The difference between the constructions is in how the buckets are sized, and how the distributor and the other functions involved are represented.

5.1 Longest Common Prefixes

The first solution, described in [5], is based on *longest common prefixes*. Let b be a positive integer (to be decided later), and divide the set S into buckets B_i

⁶ In order to highlight better the differences between the various approaches, in this section I consider the long-keys scenario.

of size⁷ b , preserving order. Observe that ℓ in this case is easy to implement, because $\ell(i) = b \cdot i$.

To build the distributor $f : S \rightarrow [m]$ (with $m = \lceil n/b \rceil$), we first observe that the longest common prefix of the keys in S_i are all different. In fact, suppose that p_i is the longest common prefix of the keys of S_i : then, either $S_i = \{p_i\}$ (in which case, p_i is clearly *not* the longest common prefix of any other bucket), or there are $v, v' \in S_i$ such that $p_i 0$ is a prefix of v and $p_i 1$ is a prefix of v' (otherwise, p_i would not be “the longest”). Suppose that $i < j$ and $p_i = p_j = p$: then a key prefixed by $p_j 0 = p 0$ would come after a key prefixed by $p_i 1 = p 1$, contradicting the fact that the buckets respect the lexicographic order of keys.

Now, let $P = \{p_0, \dots, p_{m-1}\}$ be the set of the m longest common prefixes. Let us store:

- The function $f_0 : S \rightarrow [\log u]$ mapping each key to the length of the longest common prefix of the bucket the key belongs to;
- The function $f_1 : P \rightarrow [m]$ mapping each longest common prefix to the index of the bucket it corresponds.

The distributor f can now be defined as

$$f(x) = f_1(x[: f_0(x)])$$

where $x[: t]$ is the prefix of length t of x . Both f_0 and f_1 are stored as MWHC functions, and so are the m MMPHF's $g_i : U \rightarrow [b]$.

Storing f_0 requires $(\gamma + \log \log u)n + o(n)$ bits; storing f_1 requires $(\gamma + \log m)m + o(m)$ bits; each function g_i requires $(\gamma + \log b)b + o(b)$ bits. If we let $b = \log n$ (hence $m = n/\log n$), we obtain

$$(\gamma + \log \log u)n + \frac{\gamma + \log n - \log \log n}{\log n}n + \frac{n}{\log n}(\gamma + \log \log n) \log n + o(n)$$

and since $n = O(u)$, the space required is $O(\log \log u)$ bits/key.

The function f_0 is asymptotically the most space-consuming data structure. This observation also implies that there is no point in trying to modify the way in which the g_i 's are represented (e.g., using a recursive approach).

5.2 Z-Fast Tries

The second approach to MMPHF construction that I want to present is based on z-fast tries, defined in [5], and later studied in [9, 24, 25]. This time, the construction of the distributor $f : U \rightarrow [m]$ is based on the following observation. Divide again the set S into m buckets, S_0, S_1, \dots, S_{m-1} , and let d_i be the largest (lexicographically) element of S_i , called the i -th *delimiter*. The set $D = \{d_0, \dots, d_{m-1}\}$ of delimiters can be used to build f , reducing the computation of f to a relative ranking problem: given $x \in S$, $f(x)$ is the number of $d \in D$ such that $d < x$. (We call it “relative ranking” because we only care about elements of S ; on the rest of the universe f may return any value). Now, in order to build f we will use a trie-like structure, that I will sketch below.

⁷ The last bucket may, of course, be smaller than b .

Generalities on Tries. A *binary (compact) trie* is a binary tree whose nodes are labelled with binary strings; let C_γ be the label of node γ (called the *content* of γ). The *name* N_γ of a node γ is defined recursively as follows:

- The name of the root is ϵ ;
- If ξ is the left (right, resp.) child of γ , its name is $N_\gamma C_\gamma 0$ ($N_\gamma C_\gamma 1$, resp.).

The *extent* E_γ of a node γ is $N_\gamma C_\gamma$. Given a set of binary strings D (in our case, the set of delimiters), we let $\text{Trie}(D)$ be the unique trie such that D is the set of the extents of its leaves. From now on, I will refer to this trie.

Given any binary string x , the *exit node of x* is the node whose name is the longest node name that is a prefix of x ; I write $\text{exit}(x)$ for the name of the exit node. We say that x *exits on the left* if at the first mismatch between x and the extent of its exit node, x contains a 0; otherwise, we say it *exits on the right*. By convention, elements of D exit on the left. I write $\text{exitsRight}(x)$ for the function returning 1 if $x \in S$ exits on the right, 0 if it exits on the left.

How the Distributor is Realized (I). For the moment, suppose that one is able to compute, for every $x \in S$, the exit node and direction of x , i.e., $\text{exit}(x)$ and $\text{exitsRight}(x)$. Suppose further that for every node name n one is able to compute the number of leaves on the left ($\text{leftCount}(n)$) and below ($\text{belowCount}(n)$) that node. Then, the distributor function f can be computed as

$$f(x) = \text{leftCount}(\text{exit}(x)) + \text{exitsRight}(x) \cdot \text{belowCount}(\text{exit}(x)).$$

The computation of $\text{exit}(x)$ will be the hardest part, so we first describe the remaining ingredients. The $\text{exitsRight}(-)$ function is a 1-bit function, that can be represented using the MWHC technique, so it requires $\gamma = O(1)$ bits/key. The $\text{leftCount}(-)$ and $\text{belowCount}(-)$ functions can be implemented as follows: let $X = \{N_\gamma, N_\gamma^+ \mid \gamma \text{ a node}\}$, where N_γ^+ is defined to be the bit sequence representing the successor⁸ of N_γ and having the same length as N_γ (for example, if $N_\gamma = 10011$ then $N_\gamma^+ = 10100$). Of course, $|X| = O(m)$; build a MMPHF $\xi : X \rightarrow [|X|]$, and construct a bit-array of size $|X|$ that contains a 1 in the positions of the name of leaves. It is immediate to see that, for every node γ

$$\begin{aligned} \text{leftCount}(N_\gamma) &= \text{rank}(\xi(N_\gamma)) \\ \text{belowCount}(N_\gamma) &= -\text{leftCount}(N_\gamma) + \begin{cases} \text{rank}(\xi(x1)) & \text{if } N_\gamma = x0 \\ \text{rank}(\xi(x1^+)) & \text{if } N_\gamma = x1. \end{cases} \end{aligned}$$

Overall this data structure can be represented as an LCP-based MMPHF, requiring $O(m \log \log u)$ bits (to represent ξ), plus $O(m)$ bits for the bit-array and the corresponding rank structure.

⁸ If N_γ is a sequence of 1s, N_γ^+ will not be added to the set.

How the Distributor is Realized (II). What we still have to explain is how the function $\text{exit}(-)$ is implemented. One trivial solution would be to actually store $\text{Trie}(D)$ and navigate it to determine the exit node for the input string x , but this approach would require too much space and would impose a $|x| = O(\log u)$ query time. We shall rather use an alternative approach, storing a much simpler data structure (a *z-fast trie*) that contains enough information to navigate the trie; on this structure we will proceed with a procedure that is similar to a binary search, and is called *fat binary search*.

Given an integer interval (i.e., a finite set of consecutive natural numbers) $[\ell ..r]$, call *2-fattest* the (unique) integer in the set that is divisible by the largest possible power of 2. For every node γ of $\text{Trie}(D)$, the *handle* H_γ of γ is the prefix of E_γ whose length is the 2-fattest element of $[|N_\gamma| ..|E_\gamma|]$.

The z-fast trie associated to $\text{Trie}(D)$ is a map z that maps handles to pairs: the handle H_γ is mapped to the pair $(|E_\gamma|, \sigma(E_\gamma))$ where $\sigma(-)$ is an s -bit digesting function.

In order to determine $\text{exit}(x)$ for a given input x , we can compute the largest i such that $x[:i]$ is the name of a node or, equivalently⁹ the largest i such that $x[:i]$ is the extent of a node (the name of the exit node will thus be $x[:i+1]$).

We keep track of an interval $[\ell ..r]$ that contains the index i we are looking for; at the beginning $[\ell ..r] = [1 ..|x| - 1]$. At every step, we take the 2-fattest element t of $[\ell ..r]$, and query z with $x[:t]$. Let $z(x[:t]) = (u, \gamma)$; if $u \leq |x|$ and $\sigma(x[:u]) = \gamma$, then we let $\ell = u + 1$; otherwise, we let $r = t - 1$. When $\ell \geq r$, the index i we were looking for is $i = \ell - 1$.

Fat binary search resembles standard binary search (and, like binary search, it takes $O(\log |x|) = O(\log \log u)$ steps); it is based on the following easy property: if t is the 2-fattest number in $[\ell ..r]$, and $\ell \leq \ell' \leq t \leq r' \leq r$, then t is also 2-fattest in $[\ell' ..r']$.

The map z requires $O(m(s + \log \log u))$ bits where s is the size of the signatures; observe that choosing s too small yields a high probability of false positives (sometimes you may believe that the $x[:u]$ is an extent, while it is not; this happens when $\sigma(x[:u])$ happens to coincide with the signature of an extent).

If one lets $s = \log(n/m)$, the size of the z-trie becomes $O(m(\log(n/m) + \log \log u))$; the probability of an error is $O(m/n)$, hence the expected number of errors (elements of S that are misclassified) is $O(m)$. Repeating the construction $O(1)$ times (in expectation), you obtain a $O(m)$ worst-case guarantee on the number of errors. Using the techniques described in [5, Sect. 5], we can store this misclassified set in an explicit dictionary that uses the same amount of space as z , along with a map that stores the correct output for this set of errors.

This way, the z-fast trie needs space $O(m(\log(n/m) + \log \log u))$ and has query time $O(\log \log u)$. Letting $m = n/\log \log u$, the z-fast trie requires $O(n)$ bits of space (i.e., constant number of bits/key). Also the ranking data structure for leaf-counting takes $O(m \log \log u) = O(n)$ bits of space. We finally have to

⁹ If $\text{exit}(x)$ is the root, the algorithm will return $i = -1$, so it is still true that $x[:i+1]$ is the name of the exit node.

Table 1. Time and space complexities for the data structures presented in this paper.

		Space bits/key	Access time	
			short keys	long keys
r -bit functions (4.1)		$\min(r + \gamma, r\gamma) + o(1)$	$O(1)$	$O((\log u)/w)$
MPH	MWHC (4.1)	$2\gamma + o(1) (\approx 2.65)$	$O(1)$	$O((\log u)/w)$
	HDC (4.2)	$O(1) (\approx 2.05)$	$O(1)$	$O((\log u)/w)$
MMPH	LCP (5.1)	$O(\log \log u)$	$O(1)$	$O((\log u)/w)$
	z-fast tr. (5.2)	$O(\log \log \log u)$	$O(\log \log u)$	$O((\log u)/w + \log \log u)$

represent the g_i : each of them requires $(\gamma + \log \log \log u) \log \log u + o(n)$ bits, and to store $m = n / \log \log u$ of them we need $O(\log \log \log u)$ bits/key.

6 Conclusions

In Table 1 you can find a summary of the constructions I presented, so that the different available trade-offs are more easily understood. This paper is not intended to be an exhaustive survey of the existing techniques for the construction of minimal and monotone minimal perfect hash functions: the reader that is interested in a more general presentation should for example start from [9, 22]. My aim was rather to cherry-pick from the large body of literature on this topics those constructions that I find particularly relevant, using two guiding principles: my personal taste and the fact that I believe this is one of the situations where the *techniques* are more interesting than the actual *tools*. I hope that even the occasional reader has found some inspiration (or at least some pleasure) from the algorithms I presented.

Acknowledgements. I want to thank Sebastiano Vigna for his comments and insightful suggestions. This paper is partially funded by the Google Focused Award “Web Algorithmics for Large-Scale Data Analysis”.

References

1. Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with $O(1)$ worst case access time. *J. Assoc. Comput. Mach.* **31**, 538–544 (1984)
2. Fredman, M.L., Komlós, J.: On the size of separating systems and families of perfect hash functions. *SIAM J. Algebr. Discret. Methods* **5**, 61–68 (1984)
3. Fox, E.A., Chen, Q.F., Daoud, A.M., Heath, L.S.: Order-preserving minimal perfect hash functions and information retrieval. *ACM Trans. Inf. Sys.* **9**, 281–308 (1991)
4. Majewski, B.S., Wormald, N.C., Havas, G., Czech, Z.J.: A family of perfect hashing methods. *Comput. J.* **39**, 547–554 (1996)

5. Belazzougui, D., Boldi, P., Pagh, R., Vigna, S.: Monotone minimal perfect hashing: Searching a sorted table with $O(1)$ accesses. In: Proceedings of the 20th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA), pp. 785–794, New York, ACM Press (2009)
6. Boldi, P., Vigna, S.: The WebGraph framework i: compression techniques. In: Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004), pp. 595–601, Manhattan, USA, ACM Press (2004)
7. Boldi, P., Rosa, M., Santini, M., Vigna, S.: Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In: Srinivasan, S., Ramamritham, K., Kumar, A., Ravindra, M.P., Bertino, E., Kumar, R. (eds.) Proceedings of the 20th International Conference on World Wide Web, pp. 587–596. ACM (2011)
8. Baeza-Yates, R.A., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley Longman Publishing Co. Inc., Boston (1999)
9. Belazzougui, D., Boldi, P., Pagh, R., Vigna, S.: Theory and practise of monotone minimal perfect hashing. In: Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments (ALENEX), pp. 132–144. SIAM (2009)
10. Knuth, D.E.: The Art of Computer Programming. Addison-Wesley, Boston (1973)
11. Mitzenmacher, M., Vadhan, S.: Why simple hash functions work: exploiting the entropy in a data stream. In: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, pp. 746–755. Society for Industrial and Applied Mathematics, Philadelphia (2008)
12. Jacobson, G.: Space-efficient static trees and graphs. In: 30th Annual Symposium on Foundations of Computer Science (FOCS 1989), pp. 549–554. IEEE Computer Society Press, Research Triangle Park, North Carolina (1989)
13. Patrascu, M.: Succincter. In: 49th Annual IEEE Symposium on Foundations of Computer Science, pp. 305–313. IEEE Computer Society (2008)
14. Vigna, S.: Broadword implementation of rank/select queries. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 154–168. Springer, Heidelberg (2008)
15. Gog, S., Petri, M.: Optimized succinct data structures for massive data. Software: Practice and Experience (2014). To appear
16. Bloom, B.H.: Space-time trade-offs in hash coding with allowable errors. Commun. ACM **13**, 422–426 (1970)
17. Mehlhorn, K.: Data Structures and Algorithms 1: Sorting and Searching. EATCS monographs on theoretical computer science, vol. 1. Springer, Heidelberg (1984)
18. Hagerup, T., Tholey, T.: Efficient minimal perfect hashing in nearly minimal space. In: Ferreira, A., Reichel, H. (eds.) STACS 2001. LNCS, vol. 2010, pp. 317–326. Springer, Heidelberg (2001)
19. Chazelle, B., Kilian, J., Rubinfeld, R., Tal, A.: The Bloomier filter: an efficient data structure for static support lookup tables. In: Munro, J.I. (ed.) Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, pp. 30–39. SIAM (2004)
20. Belazzougui, D., Botelho, F.C., Dietzfelbinger, M.: Hash, displace, and compress. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 682–693. Springer, Heidelberg (2009)
21. Molloy, M.: Cores in random hypergraphs and Boolean formulas. Random Struct. Algorithms **27**, 124–135 (2005)
22. Dietzfelbinger, M.: Design strategies for minimal perfect hash functions. In: Hromkovič, J., Kráľovič, R., Nunkesser, M., Widmayer, P. (eds.) SAGA 2007. LNCS, vol. 4665, pp. 2–17. Springer, Heidelberg (2007)

23. Fredriksson, K., Nikitin, F.: Simple compression code supporting random access and fast string matching. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 203–216. Springer, Heidelberg (2007)
24. Belazzougui, D., Boldi, P., Pagh, R., Vigna, S.: Theory and practice of monotone minimal perfect hashing. *ACM J. Exp. Algorithmic* **16**, 3.2:1–3.2:26 (2011)
25. Belazzougui, D., Boldi, P., Pagh, R., Vigna, S.: Fast prefix search in little space, with applications. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part I. LNCS, vol. 6346, pp. 427–438. Springer, Heidelberg (2010)

Equational Properties of Fixed Point Operations in Cartesian Categories: An Overview

Zoltán Ésik^(✉)

Department of Computer Science, University of Szeged, Szeged, Hungary
ze@inf.u-szeged.hu

Abstract. Several fixed point models share the equational properties of iteration theories, or iteration categories, which are cartesian categories equipped with a fixed point or dagger operation subject to certain axioms. After discussing some of the basic models, we provide equational bases for iteration categories and offer an analysis of the axioms. Although iteration categories have no finite base for their identities, there exist finitely based implicational theories that capture their equational theory. We exhibit several such systems. Then we enrich iteration categories with an additive structure and exhibit interesting cases where the interaction between the iteration category structure and the additive structure can be captured by a finite number of identities. This includes the iteration category of monotonic or continuous functions over complete lattices equipped with the least fixed point operation and the binary supremum operation as addition, the categories of simulation, bisimulation, or language equivalence classes of processes, context-free languages, and others. Finally, we exhibit a finite equational system involving residuals, which is sound and complete for monotonic or continuous functions over complete lattices in the sense that it proves all of their identities involving the operations and constants of cartesian categories, the least fixed point operation and binary supremum, but not involving residuals.

1 Introduction

The semantics of recursion and iteration is usually captured by fixed points of functions, functors, or other constructors. Fixed point operations have been widely used in several branches of computer science including automata and formal language theory and its generalizations, the semantics of programming languages, abstract data types, process algebra and concurrency, rewriting, programming logics and verification, complexity theory, and many other fields. The study of the equational properties of fixed point operations was initiated in the late 1960's, see [7, 25–27, 63, 65, 66, 68, 73] for a sampling of some early references.

Iteration theories were introduced in 1980 in [9] and [27]. (In [27], they were called generalized iterative theories.) The results obtained until the mid 1990's

Z. Ésik—Partially supported by grant no. ANN 110883 from the National Foundation for Scientific Research of Hungary.

(summarized in [12, 15]) indicated that they give a full account of the equational properties of most fixed point operations used in computer science.

The aim of this paper is to provide an overview of some old (see e.g. [27, 32–34]) and some recent results (see e.g. [39, 41, 42]). Unlike in [12], instead of Lawvere theories [55, 62], here we base our treatment on the slightly more general cartesian categories [5] and many-sorted theories [72]. There are several other alternative formalisms such as *abstract clones* [23], or the functional languages of μ -terms [4], or ‘*let rec expressions*’, etc.

The paper is organized as follows. In Sect. 2, we define some of the basic fixed point models such as the cartesian categories of monotonic or continuous functions of complete partial orders (cpo’s) or complete lattices, or the categories of complete metric spaces and contractions, trees and regular trees, etc. Our models will be equipped with a parametrized fixed point operation, also called dagger, such as the least or the unique fixed point operation. The main result of this section shows that our basic models satisfy the very same set of identities involving the operations and constants of cartesian categories and the fixed point operation. These identities define iteration theories, or iteration categories. In Sect. 3, we give axioms for iteration categories, and in Sect. 4, we offer an analysis of the axioms. Although there is no finite base of identities for iteration categories, in Sect. 5 we exhibit several finite axiom systems involving identities and implications (quasi-identities) that capture the equational theory of iteration categories. In Sect. 6, we enrich some of our models with an additive structure. The main results indicate that in many models of computational interest, the interaction between the iteration category structure and the additive structure can be described by a finite number of additional identities. This holds for example for the cartesian categories of monotonic or continuous functions of complete lattices with the least fixed point operation and the binary pointwise supremum operation as addition. Finally, in Sect. 7, we add residuation to the operations. We recall a recent result showing that there is a finite system of identities involving residuals, which is sound in the standard models of complete lattices and monotonic or continuous functions and complete for the identities not involving residuation.

Some Notation. We will denote the composition of morphisms $f : A \rightarrow B$ and $g : B \rightarrow C$ in a category \mathcal{C} by $g \circ f : A \rightarrow C$ and the identity morphism corresponding to an object A by id_A . For a nonnegative integer $n \in \mathbb{N}$, we let $[n]$ denote the set $\{1, \dots, n\}$, so that $[0]$ is the empty set. When X is a set, we denote by X^* the set of all finite sequences or words over X including the empty sequence ϵ .

2 Models

All categories \mathcal{C} considered in this paper will be *cartesian categories* [5]. Thus we require the existence of a *product diagram*

$$\pi_i^{C_1 \times \dots \times C_n} : C_1 \times \dots \times C_n \rightarrow C_i, \quad i \in [n],$$

for any family of objects C_i , $i \in [n]$, $n \geq 0$, with the usual universal property. When $f_i : A \rightarrow C_i$, $i \in [n]$, is a family of morphisms in \mathcal{C} , the unique mediating morphism $f : A \rightarrow C_1 \times \cdots \times C_n$ with $\pi_i^{C_1 \times \cdots \times C_n} \circ f = f_i$ for all $i \in [n]$ will be denoted $\langle f_1, \dots, f_n \rangle$. The morphism f is called the (*target*) *tupling* of the f_i . When $n = 0$, the tupling f is the unique morphism $!_A : A \rightarrow T$, where T is a fixed terminal object. It holds that

$$\text{id}_{C_1 \times \cdots \times C_n} = \langle \pi_1^{C_1 \times \cdots \times C_n}, \dots, \pi_n^{C_1 \times \cdots \times C_n} \rangle$$

for all objects C_1, \dots, C_n , $n \geq 0$, i.e., the identity morphisms are tuplings of projections. A *Lawvere theory* [62] is a cartesian category whose objects are the nonnegative integers such that each object n is the n -fold product of the generating object 1.

Our cartesian categories \mathcal{C} will be equipped with a *dagger operation* mapping a morphism $f : A \times C \rightarrow A$ to a morphism $f^\dagger : C \rightarrow A$. We list some of the categories of our interest.

1. **CPO_m** (resp. **CPO_c**) is the category of *cpo*'s and monotonic (resp. continuous) functions, see e.g. [24, 53]. The dagger operation is the (parametrized) least fixed point operation. In more detail, if $f : A \times B \rightarrow A$ is monotonic, where A, B are *cpo*'s, then for each $y \in B$, $f^\dagger(y)$ is the least (pre-)fixed point of the monotonic function $f_y : A \rightarrow A$ defined by $f_y(x) = f(x, y)$ for all $x \in A$. It is known that $f^\dagger : B \rightarrow A$ is also monotonic, and when f is continuous, then so is f^\dagger .
2. **CL_m** (resp. **CL_c**), the category of *complete lattices* and monotonic (resp. continuous) functions. The dagger operation is again the (parametrized) least fixed point operation.
3. Let **CM** denote the cartesian category of all *pointed complete metric spaces* and contractions (we only consider metric spaces such that the distance of any two points is at most 1.) Thus, when $M = (M, x_0, d)$ and $M' = (M', x'_0, d')$ are pointed complete metric spaces and $f : M \rightarrow M'$ in **CM**, then $d'(f(x), f(y)) \leq d(x, y)$ holds for all $x, y \in M$. The product of a family of metric spaces is equipped with the usual maximum distance.

Regarding **CM**, we introduce only a partial dagger operation. However, this partial dagger operation will still be sufficient for our purposes. The partial dagger operation is given as follows. Suppose that $f : A \times B \rightarrow A$ in **CM**, where A and B are pointed complete metric spaces with distinguished points x_0 and y_0 . Then f^\dagger is defined iff $\lim_{n \rightarrow \infty} f_y^n(x_0)$ exists for all $y \in B$. Moreover, in this case, $f^\dagger(y) = \lim_{n \rightarrow \infty} f_y^n(x_0)$ for all y . It is not difficult to prove that when f^\dagger is defined, then it is a contraction $B \rightarrow A$. Also, when f_y is a *proper contraction*, then $f^\dagger(y)$ is the unique fixed point of f_y by Banach's theorem.

4. Suppose that S is a set of *sorts* and Σ is an S -sorted *signature*, so that it contains a set of symbols $\Sigma_{u,s}$ for each $u \in S^*$ and $s \in S$. The objects of the category **Tree_Σ** of Σ -trees are all words in S^* . A morphism $u \rightarrow s$ ($u = s_1 \dots s_n \in S^*$, $s \in S$) is a partial Σ -tree of sort s in the sorted variables $X_u = \{x_1^u, \dots, x_n^u\}$. Such a tree may be seen as a labeled digraph, or as a

partial function $\mathbb{N}^* \rightarrow (\Sigma \cup X_u)^*$, subject to certain properties, cf. [53, 72]. When $u, v \in S^*$ with $s = s_1 \dots s_n$, a morphism $u \rightarrow v$ is an n -tuple $f = (f_1, \dots, f_n)$, where each f_i is a tree $u \rightarrow s_i$.

Composition is given by substitution for the variables and categorical product corresponds to concatenation on objects. The projection morphisms are the trees with a single node, labeled by a variable. Trees $u \rightarrow v$ may be equipped with either a complete partial order [53], or a complete metric [3] and a distinguished point, the undefined tree. When $f : u \times v \rightarrow u$, then f induces a function over $\mathbf{Tree}_\Sigma(v, u)$ defined by $g \mapsto f \circ \langle g, \text{id}_v \rangle$. The dagger operation returns the least fixed point of this function, or alternatively, we may define f^\dagger using the complete metric as for **CM**.

5. Consider the model \mathbf{Tree}_Σ and let $u, v \in S^*$ and $s \in S$. As usual, call a tree $f : u \rightarrow s$ *regular* if it has a finite number of subtrees, cf. [53]. Moreover, when $f : u \rightarrow v$, then call f regular if its components are. It is known that regular trees form a cartesian subcategory of \mathbf{Tree}_Σ closed under dagger. We denote this category by \mathbf{Reg}_Σ .

We now define terms that will be used to denote morphisms in our models. Suppose that S_0 is a set of *basic types* (or *sorts*) a, b, c, \dots . The set of *product types* consists of all finite, possibly empty sequences u, v, w, \dots in S_0^* . We usually denote such a sequence u by $a_1 \times \dots \times a_n$, where $a_i \in S_0$, for all $i \in [n]$, and call the integer $|u| = n$ the length of u . Moreover, when $u = a_1 \times \dots \times a_m$ and $v = b_1 \times \dots \times b_n$, we let $u \times v = a_1 \times \dots \times a_m \times b_1 \times \dots \times b_n$.

For each $u, v \in S_0^*$, let $\Delta_{u,v}$ be a countably infinite set of ‘morphism variables’. Then the sets $\text{TERM}_{u,v}$ of (*typed*) *dagger terms*, where $u, v \in S_0^*$, are defined by:

1. $\pi_i^u \in \text{TERM}_{u, a_i}$ for all $u = a_1 \times \dots \times a_n \in S_0^*$ and $i \in [n]$.
2. $f \in \text{TERM}_{u,v}$ for all $f \in \Delta_{u,v}$, $u, v \in S_0^*$.
3. If $t \in \text{TERM}_{u,v}$ and $t' \in \text{TERM}_{v,w}$, where $u, v, w \in S_0^*$, then $t' \circ t \in \text{TERM}_{u,w}$.
4. If $t_i \in \text{TERM}_{u, a_i}$ for all $i \in [n]$, $n \geq 0$, where $u \in S_0^*$ and $a_i \in S_0$ for all $i \in [n]$, then $\langle t_1, \dots, t_n \rangle \in \text{TERM}_{u,v}$, where $v = a_1 \times \dots \times a_n$.
5. If $t \in \text{TERM}_{u \times v, u}$, then $t^\dagger \in \text{TERM}_{v, u}$, where $u, v \in S_0^*$.

Below, instead of $t \in \text{TERM}_{u,v}$, we will usually write $t : u \rightarrow v$.

When $u, v \in S_0^*$ with $u = a_1 \times \dots \times a_m$ and $v = b_1 \times \dots \times b_n$, we define $\pi_u^{u \times v} = \langle \pi_1^{u \times v}, \dots, \pi_m^{u \times v} \rangle : u \times v \rightarrow u$ and $\pi_v^{u \times v} = \langle \pi_{m+1}^{u \times v}, \dots, \pi_{m+n}^{u \times v} \rangle : u \times v \rightarrow v$. Sometimes we will also write $\pi_{(1)}^{u \times v}$ for $\pi_u^{u \times v}$ and $\pi_{(2)}^{u \times v}$ for $\pi_v^{u \times v}$. More generally, we define $\pi_{(i)}^{u_1 \times \dots \times u_n}$ for each $u_1, \dots, u_n \in S_0^*$ and $i \in [n]$ in the same way. We will abbreviate the term $(\pi_u^{u \times v})^\dagger$ by $\perp_{v,u}$.

Suppose that $u = a_1 \times \dots \times a_m$ and $\rho : [n] \rightarrow [m]$. Let $v = a_{\rho(1)} \times \dots \times a_{\rho(n)}$. Then we associate with ρ (and u) the *base term*

$$\langle \pi_{\rho(1)}^u, \dots, \pi_{\rho(n)}^u \rangle : u \rightarrow v,$$

also denoted ρ . When $m = n$ and ρ is a bijective function, we call it a *base permutation term*. In particular, $\langle \pi_1^u, \dots, \pi_m^u \rangle$ is a base permutation term that

we denote by id_u . The *inverse* ρ^{-1} of a base permutation term $u \rightarrow u$ associated with a bijection $\rho : [n] \rightarrow [n]$ with respect to $u \in S_0^*$ is the term associated with the inverse ρ^{-1} of ρ .

For any terms $t : u \rightarrow v$ and $t' : u \rightarrow w$, where v and w have length m and n , resp., we write $\langle t, t' \rangle$ as an abbreviation for

$$\langle \pi_1^v \circ t, \dots, \pi_m^v \circ t, \pi_1^w \circ t', \dots, \pi_n^w \circ t' \rangle : u \rightarrow v \times w.$$

Moreover, when $t : u \rightarrow v$ and $t' : u' \rightarrow v'$, we define $t \times t' : u \times u' \rightarrow v \times v'$ as the term $\langle t \circ \pi_u^{u \times u'}, t' \circ \pi_{u'}^{u \times u'} \rangle$.

When \mathcal{C} is a cartesian category with a dagger operation and each basic type $a \in S_0$ is interpreted as an object $A = \llbracket a \rrbracket$ of \mathcal{C} , then each $u = a_1 \times \dots \times a_n \in S_0^*$ can naturally be interpreted as the object $A_1 \times \dots \times A_n$, where $A_i = \llbracket a_i \rrbracket$ for all $i \in [n]$. In particular, $\llbracket \epsilon \rrbracket$ is a fixed terminal object T (see also above). We assume that $\llbracket uv \rrbracket = \llbracket u \rrbracket \times \llbracket v \rrbracket$, for all $u, v \in S_0^*$.

And if each $f \in \Delta_{u,v}$, $u, v \in S_0^*$, is interpreted as a morphism $\llbracket u \rrbracket \rightarrow \llbracket v \rrbracket$ in \mathcal{C} , then each term $t \in \text{TERM}_{u,v}$, $u, v \in S_0^*$, can be interpreted as a morphism $\llbracket t \rrbracket : \llbracket u \rrbracket \rightarrow \llbracket v \rrbracket$ in \mathcal{C} . The constant π_i^u , where $u = a_1 \times \dots \times a_n$ and $i \in [n]$, is interpreted as the projection $\llbracket a_1 \rrbracket \times \dots \times \llbracket a_n \rrbracket \rightarrow \llbracket a_i \rrbracket$ according to a selected product diagram. We will assume that the selected product diagrams are ‘associative on the nose’. In particular, we assume that when $n = 1$ and $i = 1$, then the projection π_i^u is the appropriate identity morphism. We skip the straightforward definition but remark that when t is a base term, its interpretation is a tupling of projections, called a *base morphism*.

Suppose that t and t' are in $\text{TERM}_{u,v}$. Then we say that the formal equality $t = t'$ is an *identity*. Let \mathcal{C} be a cartesian category with a dagger operation. We say that an identity $t = t'$ *holds in* \mathcal{C} , or *is satisfied by* \mathcal{C} , or *is valid in* \mathcal{C} , if $\llbracket t \rrbracket = \llbracket t' \rrbracket$ holds for all interpretations over \mathcal{C} . The *equational theory* of \mathcal{C} is the set of all identities satisfied by \mathcal{C} .

The following theorem summarizes several results treated in [12]. (Regarding **CM**, we only consider those interpretations that assign to each $f \in \Delta_{u,v}$ a function which is a tupling of proper contractions and projections, and the projections are assumed to preserve the distinguished points.)

Theorem 1. *The categories **CPO**_m, **CPO**_c, **CL**_m, **CL**_c, **CM**, **Tree**_Σ, **Reg**_Σ satisfy the same set of identities.*

Motivated by Theorem 1, we adopt the following semantic definition of iteration categories.

Definition 1. *We define an iteration category to be a cartesian category \mathcal{C} with a dagger operation satisfying all identities that hold in **CPO**_m. An iteration theory is an iteration category which is a Lawvere theory.*

We say that a cartesian category \mathcal{C} is nontrivial if there is some object A such that $\pi_1^{A \times A} \neq \pi_2^{A \times A}$. An equivalent condition is that some hom-set $\mathcal{C}(A, B)$ has at least two morphisms. It is shown in essence in [71] that if \mathcal{C} is nontrivial cartesian category with a dagger operation which is an iteration category,

then either \mathcal{C} satisfies exactly the identities that hold in all iteration categories, or it satisfies exactly the identities satisfied by those iteration categories having a unique morphism $T \rightarrow A$ for each object A , where T is a terminal object.

Some further iteration categories or theories are the continuous and rational theories of [73], Elgot's (pointed) iterative theories [26], matrix theories over complete or inductive semirings [12, 45], theories of synchronization trees and bisimulation equivalence classes of synchronization trees [13], matricial theories of languages of finite and infinite words [12], theories of continuous functors [10] and functor theories over algebraically complete categories [49], iteration 2-theories [19], or the more recent cartesian categories of stratified complete lattices [43, 50] used to solve fixed point equations involving non-monotonic functions, or the categories of formal power series over complete semirings [48].

The fixed point operation appears in several different forms. In cartesian categories with an appropriate additive structure on the hom-sets, it may be replaced by a *generalized star operation* $f \mapsto f^*$, where $f : A \times B \rightarrow A$ and $f^* : A \times B \rightarrow A$, or sometimes even by a star operation mapping a morphism $f : A \rightarrow A$ to a morphism $f^* : A \rightarrow A$, as in matrix theories over semirings. For details, see [12]. It is known that in cartesian categories, the fixed point operation is equivalent to a certain *trace* or *feedback operation* that maps a morphism $f : A \times B \rightarrow A \times C$ to a morphism $\uparrow f : B \rightarrow C$. See [6] and [22, 54, 56], or [12].

3 Axiomatization

In the previous section, we gave a semantic definition of iteration categories. In this section, we provide equational axioms for them consisting of axioms for cartesian categories, and axioms involving dagger.

A possible set of cartesian axioms is [72]:

$$\begin{aligned} h \circ (g \circ f) &= (h \circ g) \circ f, & f : u \rightarrow v, g : v \rightarrow w, h : w \rightarrow z \\ \text{id}_v \circ f &= f = f \circ \text{id}_u, & f : u \rightarrow v \\ \pi_i^v \circ \langle f_1, \dots, f_n \rangle &= f_i, & v = a_1 \times \dots \times a_n, f_i : u \rightarrow a_i, i \in [n] \\ \langle \pi_1^v \circ f, \dots, \pi_n^v \circ f \rangle &= f, & f : u \rightarrow v, |v| = n \\ \pi_1^a &= \langle \pi_1^a \rangle \end{aligned}$$

Any model of these identities satisfies $g = \langle g \rangle$, where $g : u \rightarrow a$. Also, the following hold, for appropriate f, g, h, k :

$$\begin{aligned} \langle \langle f, g \rangle, h \rangle &= \langle f, \langle g, h \rangle \rangle \\ (f \times g) \times h &= f \times (g \times h) \\ (f \times g) \circ \langle h, k \rangle &= (f \circ h) \times (g \circ k) \end{aligned}$$

Below we will implicitly always assume the cartesian identities.

The *Conway identities* [12] are the *parameter* (1), *double dagger* (2) and *composition identities* (3) given below.

$$(f \circ (\text{id}_u \times g))^\dagger = f^\dagger \circ g, \quad f : u \times v \rightarrow u, \quad g : w \rightarrow v \quad (1)$$

$$(f \circ (\langle \text{id}_u, \text{id}_u \rangle \times \text{id}_v))^\dagger = f^{\dagger\dagger}, \quad f : u \times u \times v \rightarrow u \quad (2)$$

$$(f \circ \langle g, \pi_w^{u \times w} \rangle)^\dagger = f \circ \langle (g \circ \langle f, \pi_w^{v \times w} \rangle)^\dagger, \text{id}_w \rangle \quad (3)$$

where in the last identity, $f : v \times w \rightarrow u$, $g : u \times w \rightarrow v$.

Definition 2. A Conway category is a cartesian category equipped with a dagger operation satisfying the Conway identities.

The terminology follows [12] and is due to the form of these identities in matrix theories. Note that in Conway categories, the *fixed point identity*

$$f^\dagger = f \circ \langle f^\dagger, \text{id}_v \rangle, \quad f : u \times v \rightarrow u \quad (4)$$

is a particular instance of the composition identity, and the identity

$$(f \circ \pi_v^{u \times v})^\dagger = f, \quad f : v \rightarrow u \quad (5)$$

is a particular instance of the fixed point identity (4), while

$$(f \circ \pi_{u \times v}^{u \times v \times w})^\dagger = f^\dagger \circ \pi_v^{v \times w}, \quad f : u \times v \rightarrow u \quad (6)$$

is an instance of the parameter identity (1). The parameter, double dagger and composition identities are sometimes called the identities of *naturality*, *diagonality* and *dinaturality*, see e.g. [71].

Conway categories satisfy several other non-trivial identities including the *Bekić identity* [7, 25] (called the *pairing identity* in [12])

$$\langle f, g \rangle^\dagger = \langle f^\dagger \circ \langle h^\dagger, \text{id}_w \rangle, h^\dagger \rangle \quad (7)$$

or its ‘dual form’

$$\langle f, g \rangle^\dagger = \langle k^\dagger, \bar{g}^\dagger \circ \langle k^\dagger, \text{id}_w \rangle \rangle, \quad (8)$$

where $f : u \times v \times w \rightarrow u$ and $g : u \times v \times w \rightarrow v$ and

$$\begin{aligned} h &= g \circ \langle f^\dagger, \text{id}_{v \times w} \rangle : v \times w \rightarrow v \\ k &= f \circ \langle \pi_u^{u \times w}, \bar{g}^\dagger, \pi_w^{u \times w} \rangle : u \times w \rightarrow u \end{aligned}$$

with $\bar{g} = g \circ (\langle \pi_v^{u \times v}, \pi_u^{u \times v} \rangle \times \text{id}_w) : v \times u \times w \rightarrow v$. We will also make use of the *permutation identity* that holds in all Conway categories:

$$(\pi \circ f \circ (\pi^{-1} \times \text{id}_v))^\dagger = \pi \circ f^\dagger, \quad (9)$$

where $f : u \times v \rightarrow u$ and $\pi : u \rightarrow u$ is a base permutation term. An alternative axiomatization of Conway categories consists of the identities (5), (6), (7) and (9). For this and related facts, we refer to [12].

The Conway identities are quite powerful, for example, a general ‘Kleene theorem’ holds in all Conway categories, and both the soundness and relative completeness of Hoare’s logic can be proved just from the Conway identities, cf. [11, 12]. However, they are not complete for iteration categories. The missing ingredient is captured by the notion of identities associated with finite automata. In order to define these identities, we need to introduce some definitions and notation.

Suppose that $\mathbf{Q} = (Q, Z)$ is a (*deterministic*) *finite automaton*, where Q is the finite nonempty set of *states*, Z is the finite nonempty set of *input letters* together with a *right action* $Q \times Z \rightarrow Q$, $(q, z) \mapsto qz$. Let $Q = \{q_1, \dots, q_n\}$ and $Z = \{z_1, \dots, z_m\}$, say. For each $i \in [n]$, let

$$\rho_i^{\mathbf{Q}} = \langle \pi_{(iz_1)}^{u^n}, \dots, \pi_{(iz_m)}^{u^n} \rangle : u^n \rightarrow u^m,$$

$u \in S_0^*$, where we identify each state q_i with the integer i . Moreover, let $f^{\mathbf{Q}} : u^n \times w \rightarrow u^n$ be the tupling of the terms $f \circ (\rho_i^{\mathbf{Q}} \times \text{id}_w) : u^n \times w \rightarrow u$, where $f : u^m \times w \rightarrow u^n$.

Definition 3. [32] *The identity $\mathbf{C}(\mathbf{Q})$ associated with \mathbf{Q} is*

$$(f^{\mathbf{Q}})^{\dagger} = \tau_n \circ (f \circ (\tau_m \times \text{id}_w))^{\dagger},$$

where $f : u^m \times w \rightarrow u$, $\tau_n : u \rightarrow u^n$ is the diagonal $\langle \text{id}_u, \dots, \text{id}_u \rangle$ and τ_m is defined similarly. Suppose that a state $q = q_i$ of \mathbf{Q} is distinguished, so that (\mathbf{Q}, q) is an initialized finite automaton. The identity $\mathbf{C}(\mathbf{Q}, q)$ associated with (\mathbf{Q}, q) is

$$\pi_{(i)}^{u^n} \circ (f^{\mathbf{Q}})^{\dagger} = (f \circ (\tau_m \times \text{id}_w))^{\dagger},$$

where $f : u^m \times w \rightarrow u$.

Suppose that \mathcal{C} is a Conway category. Since the permutation identity holds in \mathcal{C} , the validity of the identity associated with a finite automaton \mathbf{Q} does not depend on the enumeration of the states or input letters. Also, $\mathbf{C}(\mathbf{Q})$ holds in \mathcal{C} iff $\mathbf{C}(\mathbf{Q}, q)$ holds for all states q of \mathbf{Q} .

Theorem 2. [27] *The Conway identities and the identities associated with finite automata form a complete set of identities of iteration categories.*

Hence, these identities form a sound and complete axiomatization of iteration categories. In [27], the ‘commutative identities’ were used instead of the identities associated with finite automata. However, these are a very close variant of the identities associated with finite automata.

4 Analysis of the Axioms

Are all identities associated with finite automata strictly needed for completeness in Theorem 2? When is a set of identities consisting of the Conway identities

and a subcollection of the identities associated with finite automata complete? In this section, we provide an answer to this question using the Krohn-Rhodes decomposition of finite automata [51, 52, 61] and a result from [35].

Suppose that $\mathbf{Q} = (Q, Z)$ is a finite automaton. The action of Z on Q is extended in the usual way to a right action $Q \times Z^* \rightarrow Q$ of Z^* on Q . When $\alpha \in Z^*$, we call the function $Q \rightarrow Q$ defined by $q \mapsto q\alpha$ for all $q \in Q$ the *function induced by α* and denote it by $\alpha^{\mathbf{Q}}$. These functions form a (finite) monoid $M(\mathbf{Q})$ whose multiplication is given by $\alpha^{\mathbf{Q}}\beta^{\mathbf{Q}} = (\alpha\beta)^{\mathbf{Q}} = \beta^{\mathbf{Q}} \circ \alpha^{\mathbf{Q}}$, for all $\alpha, \beta \in Z^*$.

Suppose that S and S' are finite semigroups. As usual, we say that S *divides* S' , or S is a *divisor* of S' , if S is a homomorphic image of a subsemigroup of S' . It is clear that this *divisibility relation* is transitive. It is known that if a group G is a divisor of a finite semigroup S , then there is a group H in S such that G is a homomorphic image of H . Moreover, we say that a finite group G is *simple* if it is nontrivial and has no nontrivial homomorphic image other than groups isomorphic to G (or equivalently, its only nontrivial normal subgroup is G itself).

Let (\mathbf{Q}, q) be an initialized finite automaton, where $\mathbf{Q} = (Q, Z)$. We call (\mathbf{Q}, q) an *initially connected finite automaton* if $Q = \{q\alpha : \alpha \in Z^*\}$.

Below we will write $\mathcal{C} \models \mathbf{C}(\mathbf{Q})$ to mean that the cartesian category \mathcal{C} equipped with a dagger operation satisfies $\mathbf{C}(\mathbf{Q})$. We will use similar notation for initially connected finite automata (\mathbf{Q}, q) .

Theorem 3. [33, 42] *Suppose that \mathcal{Q} is a set of finite automata (initially connected finite automata, resp.). Then the Conway identities and the identities associated with the members of \mathcal{Q} form a complete set of identities for iteration categories iff for each finite simple group G there is some \mathbf{Q} in \mathcal{Q} ($(\mathbf{Q}, q) \in \mathcal{Q}$) such that G divides the monoid $M(\mathbf{Q})$ of \mathbf{Q} .*

The first part is from [33], and the second is from [42]. Each finite monoid M may be seen as a finite automaton $\mathbf{Q}_M = (M, M)$ equipped with the natural right action given by the multiplication operation of M . Hence, we may define the identity $\mathbf{C}(M)$ associated with M as the identity $\mathbf{C}(\mathbf{Q}_M)$ associated with \mathbf{Q}_M .

Corollary 1. [32] *Suppose that \mathcal{M} is a set of finite monoids. Then the Conway identities and the identities associated with the members of \mathcal{M} form a complete set of identities of iteration categories iff for each finite simple group G there is some $M \in \mathcal{M}$ such that G divides M .*

For each $n \geq 3$, consider the n -state initially connected automaton (\mathbf{Q}_n, q_1) with state set $\{q_1, \dots, q_n\}$, an input letter inducing the cyclic permutation $(q_1 \dots q_n)$ and a letter inducing the transposition $(q_1 q_2)$, so that the monoid of \mathbf{Q}_n is the symmetric group S_n of degree n . Then the Conway identities together with any infinite subsystem of the identities associated with the initially connected automata (\mathbf{Q}_n, q_1) are complete for iteration categories, since every finite group

can be embedded in any large enough symmetric group. It is shown in [33] that in Conway categories, the identity associated with (\mathbf{Q}_n, q_1) may be reduced to

$$(f \circ (\tau_2 \times \text{id}_w)) \circ \langle f \circ \langle \pi_u^{u \times w}, (f^\dagger)^{n-2}, \pi_w^{u \times w} \rangle \pi_w^{u \times w} \rangle^\dagger = (f \circ (\tau_2 \times \text{id}_w))^\dagger \quad (10)$$

where $f : u^2 \times w \rightarrow u$ and $\tau_2 = \langle \text{id}_u, \text{id}_u \rangle : u \rightarrow u^2$. (Here, $(f^\dagger)^1 = f^\dagger$ and $(f^\dagger)^m = f^\dagger \circ \langle (f^\dagger)^{m-1}, \pi_w^{u \times w} \rangle$ for all $m \geq 2$.)

Theorem 4. [33] *The Conway identities together with any infinite subcollection of the identities (10) are complete for iteration categories.*

Theorem 3 follows from Theorem 2 and the following result:

Theorem 5. [33, 42] *Suppose that \mathcal{Q} is a set of finite automata (initially connected finite automata, resp.) and \mathbf{Q} is a finite automaton. Then $\mathbf{C}(\mathbf{Q})$ holds in all Conway categories satisfying the identities associated with the members of \mathcal{Q} iff for every simple group divisor G of $M(\mathbf{Q})$ there is some $\mathbf{Q}' \in \mathcal{Q}$ ($(\mathbf{Q}', q') \in \mathcal{Q}$, resp.) such that G divides $M(\mathbf{Q}')$.*

In the rest of this section we outline a proof of one direction of Theorem 5 which, by using recent advances [42], is simpler than the one in [33].

We start by recalling the *cascade composition* of finite automata [51, 52]. Let $\mathbf{Q} = (Q, X)$ and $\mathbf{Q}' = (Q', Y)$ be finite automata. Suppose that φ is a function $Q \times X \rightarrow Y$. For each $q \in Q$ and $x \in X$, we will denote $\varphi(q, x)$ by ${}^q x$. The cascade composition of \mathbf{Q} and \mathbf{Q}' with respect to the connecting function φ is defined as the finite automaton $\mathbf{Q} \times_\varphi \mathbf{Q}' = (Q \times Q', X)$ with action $(q, q')x = (qx, q'{}^q x)$ for all $q \in Q, q' \in Q'$ and $x \in X$.

The *direct product* is a special case of the cascade composition. It is obtained by letting $X = Y$ and choosing φ to be the identity function $X \rightarrow X$. Another special case arises when \mathbf{Q} is a trivial 1-state automaton. Then φ may be viewed as a function $X \rightarrow Y$, so that the cascade composition is obtained from \mathbf{Q}' by changing the input set Y to X and defining the function induced by each letter $x \in X$ as the function induced by some letter $y \in Y$ in \mathbf{Q}' . In this special case, we call the cascade composition a *letter renaming* of \mathbf{Q}' .

Theorem 6. [32] *Suppose that \mathcal{C} is a Conway category. Let $\mathbf{Q} = \mathbf{Q}_1 \times_\varphi \mathbf{Q}_2$ be a cascade product of finite automata \mathbf{Q}_1 and \mathbf{Q}_2 . If $\mathcal{C} \models \mathbf{C}(\mathbf{Q}_2)$ then $\mathcal{C} \models \mathbf{C}(\mathbf{Q})$ iff $\mathcal{C} \models \mathbf{C}(\mathbf{Q}_1)$.*

Proposition 1. [32] *Suppose that \mathcal{C} is a Conway category. If \mathbf{Q}' is a subautomaton of a finite automaton \mathbf{Q} and $\mathcal{C} \models \mathbf{C}(\mathbf{Q})$, then $\mathcal{C} \models \mathbf{C}(\mathbf{Q}')$.*

Suppose that $\mathbf{Q} = (Q, X)$ and $\mathbf{Q} = (Q, Y)$ are finite automata with the same set of states. We say that \mathbf{Q}' is a *word renaming* of \mathbf{Q} if each function induced by a letter $y \in Y$ in \mathbf{Q}' is induced by some word $\alpha \in X^*$ in \mathbf{Q} .

Theorem 7. [42] *Suppose that \mathcal{C} is a Conway category. If a finite automaton \mathbf{Q}' is a word renaming of \mathbf{Q} , and if $\mathcal{C} \models \mathbf{C}(\mathbf{Q})$, then $\mathcal{C} \models \mathbf{C}(\mathbf{Q}')$.*

Let \mathbf{U} denote a 2-state automaton with 3 input letters, inducing the identity function and the 2 constant functions on the set of states. It is easy to see that the following holds.

Proposition 2. [32] *The identity $\mathbf{C}(\mathbf{U})$ holds in all Conway categories.*

Let \mathcal{Q} be a set of finite automata and \mathcal{C} a Conway category satisfying the identities associated with the members of \mathcal{Q} . Let \mathbf{Q} be a finite automaton such that every finite simple group dividing the monoid of some automaton in \mathcal{Q} . In order to prove the sufficiency part of Theorem 5, we need to show that $\mathcal{C} \models \mathbf{C}(\mathbf{Q})$.

By the Krohn-Rhodes decomposition theorem, \mathbf{Q} can be constructed from the automata in $\mathcal{Q} \cup \{\mathbf{U}\}$ by taking word renamings, cascade compositions, subautomata and homomorphic images. While it is obvious by induction from the above results that if \mathbf{Q} can be constructed from $\mathcal{Q} \cup \{\mathbf{U}\}$ by using only word renamings, cascade compositions and subautomata, then the identity associated with \mathbf{Q} holds in \mathcal{C} , there seems to be no simple way of extending the induction to homomorphic images. We outline a possible solution below.

If $\mathbf{Q} = (Q, Z) \in \mathcal{Q}$ with $M = M(\mathbf{Q})$, then consider the finite automaton $\mathbf{Q}' = (Q, M)$ where $qm = qu^{\mathbf{Q}}$ for all $q \in Q$ and $m = u^{\mathbf{Q}} \in M$. By Theorem 7, $\mathcal{C} \models \mathbf{C}(\mathbf{Q}')$. Consider now the monoid automaton $\mathbf{Q}_M = (M, M)$. It is well-known that \mathbf{Q}_M can be embedded in a direct power of \mathbf{Q}' . Thus, by Theorem 6, it follows that $\mathcal{C} \models \mathbf{C}(M)$. Also, if H is any group in M , then it follows using Proposition 1 (and letter renaming) that $\mathcal{C} \models \mathbf{C}(H)$. Consider a normal subgroup N of H and the automata $\mathbf{Q}_{H/N}$ and \mathbf{Q}_N associated with H/N and N , resp. We have that $\mathcal{C} \models \mathbf{C}(\mathbf{Q}_N)$. It is well-known that \mathbf{Q}_H is isomorphic to a cascade product of a letter renaming of $\mathbf{Q}_{H/N}$ and \mathbf{Q}_N , and since $\mathcal{C} \models \mathbf{C}(\mathbf{Q}_H)$ and $\mathcal{C} \models \mathbf{C}(\mathbf{Q}_N)$, it follows from Theorem 6 that $\mathcal{C} \models \mathbf{C}(\mathbf{Q}_{H/N})$. Since every simple group divisor of M divides some group H in M , and if N is a normal subgroup of H then every simple group divisor of H divides either N or H/N , it follows now by induction that \mathcal{C} satisfies the identity associated with any simple group divisor of $M = M(\mathbf{Q})$. Since $\mathbf{Q} \in \mathcal{Q}$ was arbitrary, we established that \mathcal{C} satisfies the identity associated with every simple group dividing the monoid of some automaton in \mathcal{Q} . Also, by a similar argument, we have:

Fact. If H is a finite group such that every simple group dividing H divides the monoid of some member of \mathcal{Q} , then $\mathcal{C} \models \mathbf{C}(H)$.

Call a finite automaton $\mathbf{Q} = (Q, Z)$ a *permutation automaton* if each $z \in Z$ induces a permutation of Q . It follows that $\alpha^{\mathbf{Q}}$ is a permutation of Q for all $\alpha \in Z^*$, so that $M(\mathbf{Q})$ is a group. Moreover, call $\mathbf{Q} = (Q, Z)$ a *permutation-reset automaton* if each $z \in Z$ induces a permutation or a constant map. Using the above fact, it follows as in Sects. 13 and 14 of [32] that if \mathbf{Q} is a permutation-reset automaton such that every simple group divisor of $M(\mathbf{Q})$ divides the monoid of some automaton in \mathcal{Q} , then $\mathcal{C} \models \mathbf{C}(\mathbf{Q})$. In order to complete the proof of the sufficiency part of Theorem 5, we need to establish this for all finite automata.

Suppose that $\mathbf{Q} = (Q, X)$ and $\mathbf{Q}' = (Q', X)$ are finite automata and $h : Q \rightarrow Q'$ is a surjective homomorphism $\mathbf{Q} \rightarrow \mathbf{Q}'$. Call h a *regular permutation-reset homomorphism* if the following conditions hold:

- For any two nontrivial congruence classes C, C' of $\ker(h)$ there is a word $\alpha \in X^*$ such that the restriction of $\alpha^{\mathbf{Q}}$ onto C is a bijection $C \rightarrow C'$.
- If C, C' are congruence classes of $\ker(h)$ and $\alpha \in Z^*$ such that $\alpha^{\mathbf{Q}}$ maps C into C' , then the restriction of $\alpha^{\mathbf{Q}}$ onto C is either a constant function or a bijection.

Let C be a congruence class of $\ker(h)$. Denote by $M(C)$ the monoid of all functions $C \rightarrow C$ obtained as restrictions of functions $Q \rightarrow Q$ in $M(\mathbf{Q})$. When \mathcal{G} is a set of finite simple groups, call h a \mathcal{G} -homomorphism if for any congruence class C of $\ker(h)$, every simple group divisor of $M(C)$ divides a group in \mathcal{G} . A novel proof of the Krohn-Rhodes theorem in [35] is based on the following result:

Theorem 8. [35] *Let \mathcal{G} be a set of finite simple groups and suppose that $\mathbf{Q} = (Q, X)$ and $\mathbf{Q}' = (Q', X)$ are finite automata and $h : Q \rightarrow Q'$ is a surjective \mathcal{G} -homomorphism $\mathbf{Q} \rightarrow \mathbf{Q}'$. Then there is a sequence $\mathbf{Q}_0, \dots, \mathbf{Q}_n$ of finite automata such that $\mathbf{Q}_0 = \mathbf{Q}$, $\mathbf{Q}_n = \mathbf{Q}'$ and for each $i \in [n]$, there is a regular permutation-reset \mathcal{G} -homomorphism $\mathbf{Q}_{i-1} \rightarrow \mathbf{Q}_i$ or $\mathbf{Q}_i \rightarrow \mathbf{Q}_{i-1}$.*

The next proposition is the final ingredient in our proof of the sufficiency part of Theorem 5. Let \mathcal{G} be as above.

Proposition 3. [32] *Suppose that $\mathbf{Q} = (Q, X)$ and $\mathbf{Q}' = (Q', X)$ are finite automata and $h : Q \rightarrow Q'$ is a regular permutation-reset \mathcal{G} -homomorphism $\mathbf{Q} \rightarrow \mathbf{Q}'$. Suppose that \mathcal{C} is a Conway category satisfying the identities $\mathbf{C}(G)$ associated with the members G of \mathcal{G} . Then $\mathcal{C} \models \mathbf{C}(\mathbf{Q})$ iff $\mathcal{C} \models \mathbf{C}(\mathbf{Q}')$.*

Actually this result follows from Theorem 6 by a construction showing that if h is a regular permutation-reset \mathcal{G} -homomorphism, then \mathbf{Q} can be embedded in a cascade product of \mathbf{Q}' and a permutation-reset automaton whose simple group divisors are divisors of groups in \mathcal{G} .

The proof of the sufficiency part of Theorem 5 can now be completed as follows. Suppose that \mathcal{Q} is a set of finite automata. Let \mathcal{G} contain an isomorphic copy of the simple group divisors of the monoids of the automata in \mathcal{Q} . Let \mathcal{C} be a Conway category satisfying all identities associated with the members of \mathcal{Q} . We already know that \mathcal{C} satisfies the identity of any finite permutation-reset automaton \mathbf{Q} such that any finite simple group dividing $M(\mathbf{Q})$ is isomorphic to a group in \mathcal{G} .

Now let \mathbf{Q} denote a finite automaton such that every simple group divisor of $M(\mathbf{Q})$ has an isomorphic copy in \mathcal{G} . We want to prove that $\mathcal{C} \models \mathbf{C}(\mathbf{Q})$.

Consider the unique homomorphism from \mathbf{Q} onto a trivial 1-state automaton \mathbf{Q}' . By Theorem 8, there is a finite sequence $\mathbf{Q}_0, \dots, \mathbf{Q}_n$ of finite automata such that $\mathbf{Q}_0 = \mathbf{Q}$, $\mathbf{Q}_n = \mathbf{Q}'$ and for each $i \in [n]$, there is a regular permutation-reset \mathcal{G} -homomorphism $\mathbf{Q}_{i-1} \rightarrow \mathbf{Q}_i$ or $\mathbf{Q}_i \rightarrow \mathbf{Q}_{i-1}$. It follows by repeated use of Proposition 3 that $\mathcal{C} \models \mathbf{C}(\mathbf{Q})$ iff $\mathcal{C} \models \mathbf{C}(\mathbf{Q}')$. But the latter identity is easily shown to hold in all Conway categories by repeated use of the double dagger identity.

For initially connected finite automata, the sufficiency of Theorem 5 now follows from:

Proposition 4. [42] *Suppose that (\mathbf{Q}, q) is an initially connected finite automaton and \mathcal{C} is a Conway category. If $\mathcal{C} \models \mathbf{C}(\mathbf{Q}, q)$ then $\mathcal{C} \models \mathbf{C}(\mathbf{Q})$.*

Suppose that \mathcal{G} is a set of nontrivial finite groups. Then there is a ‘variety’ $\mathcal{V}_{\mathcal{G}}$ of Conway categories corresponding to \mathcal{G} which consists of those Conway categories satisfying the identities associated with the members of \mathcal{G} . (We have $\mathcal{V}_{\mathcal{G}_1} \subseteq \mathcal{V}_{\mathcal{G}_2}$) iff every finite simple group dividing a group in \mathcal{G}_1 divides a group in \mathcal{G}_2 .) The free categories (or rather, theories) in $\mathcal{V}_{\mathcal{G}}$ have been described in [36]. In the particular case when \mathcal{G} is empty, this description agrees with the characterization of the free Conway categories [8].

5 Implicational Axiomatizations

As a corollary of Theorem 3, we obtain:

Theorem 9. [18] *There is no finite complete set of identities of iteration categories.*

For a more direct proof, we refer to [18]. Many non-finitely based equational theories have a relatively finite axiomatization with respect to iteration categories (or theories), for example the equational theory of regular languages, or bisimilarity equivalence classes of regular processes. See Sect. 6. Theorem 9 can also be derived using these relative axiomatization results.

Although the equational theory of iteration categories is not finitely based, it can be captured by a finite number of implications involving equalities, and sometimes other relations depending on the context.

Our starting point is the identities associated with finite automata. In Conway categories, these are all consequences of the *weak functoriality* implications:

$$f \circ (\tau_n \times \text{id}_w) = \tau_n \circ g \Rightarrow f^\dagger = \tau_n \circ g^\dagger, \quad n \geq 2,$$

for all $f : u^n \times w \rightarrow u^n$ and $g : u \times w \rightarrow u$, where $\tau_n : u \rightarrow u^n$ is the n -fold tupling $\langle \text{id}_u, \dots, \text{id}_u \rangle$. Indeed, let $f : u^m \times w \rightarrow u$ and $g = f \circ (\tau_m \times \text{id}_w)$, then using the notation introduced in Definition 3, we clearly have $f^{\mathbf{Q}} \circ (\tau_n \times \text{id}_w) = \tau_n \circ g$, hence $(f^{\mathbf{Q}})^\dagger = \tau_n \circ g^\dagger$, by weak functoriality. Since the weak functoriality implications hold in **CPO**, **CPO_m**, **CM**, **ΣTree** and many other models, we immediately obtain:

Proposition 5. *An identity holds in all iteration categories iff it holds in all Conway categories satisfying weak functoriality.*

The weak functoriality implications already appear in [27], where Proposition 5 is implicit. In fact, the commutative identities were introduced in [27] just in order to make one step further by replacing these implications with weaker and pure equational axioms that still guarantee completeness. The existence of an iteration theory not satisfying the weak functoriality implication for $n = 2$ was pointed out in [28]. It is shown in [29] that in Conway categories, if the weak

functoriality implications hold when u is a basic type a , then so do the generic forms of these implications. Moreover, as shown in [14], for each $n \geq 2$, there is a Conway category satisfying the weak functorial implication for all $2 \leq m \leq n$. but not satisfying this implication for $n + 1$.

Another variant of Proposition 5 concerns functoriality with respect to ‘pure’ [12] or ‘strict’ morphisms:

Proposition 6. *An identity holds in all iteration categories iff it holds in all Conway categories satisfying the following rule:*

$$f \circ (h \times \text{id}_w) = h \circ g \wedge h \circ \perp_{\epsilon, u} = \perp_{\epsilon, v} \Rightarrow f^\dagger = h \circ g^\dagger$$

for all $f : u \times w \rightarrow u$, $g : v \times w \rightarrow v$ and $h : u \rightarrow v$.

Again, this implication holds in most standard models. Thus, Proposition 6 is clear from Proposition 5 since any base morphism is pure.

There are several other similar completeness results, including e.g. the implication

$$f^{\dagger\dagger} = g^{\dagger\dagger} \Rightarrow f^{\dagger\dagger} = (f \circ \langle g^\dagger, \text{id}_{u \times w} \rangle)^\dagger,$$

where $f, g : u \times u \times w \rightarrow w$, introduced in [12] as a generalization of an axiom proposed for regular languages in [2], a version of the *Scott induction principle* [44], or the (conditional) *unique fixed point rule* of Elgot [26], cf. [12, 27].

In the ordered setting, the *fixed point induction rule* (or *least pre-fixed point rule*) also gives rise to completeness. Call a cartesian category \mathcal{C} *ordered* if each hom-set $\mathcal{C}(A, B)$ comes with a partial order \leq preserved by composition and tupling. For example, \mathbf{CPO}_m , \mathbf{CPO}_c , \mathbf{CL}_m , \mathbf{CL}_c , equipped with the pointwise order of functions, are all ordered. The cartesian categories \mathbf{Tree}_Σ and \mathbf{Reg}_Σ can also be turned into ordered cartesian categories. Each of the above models satisfies the fixed point induction rule:

$$f \circ \langle g, \text{id}_v \rangle \leq g \Rightarrow f^\dagger \leq g, \tag{11}$$

for all $f : u \times v \rightarrow u$ and $g : v \rightarrow u$.

Theorem 10. [30] *An identity holds in all iteration categories iff it holds in all ordered cartesian categories equipped with a dagger operation satisfying the fixed point (4) and parameter identities (1) and the fixed point induction rule (11).*

In all such theories, the dagger operation is also monotonic. There is another, stronger version of this result.

Theorem 11. [30] *An identity holds in all iteration categories iff it holds in all ordered cartesian categories equipped with a dagger operation satisfying the fixed point (4), parameter (1) and pairing identities (7) and a weak form of the fixed point induction rule: $f \circ \langle g, \text{id}_v \rangle = g \Rightarrow f^\dagger \leq g$, where $f : u \times v \rightarrow u$ and $g : v \rightarrow u$.*

A natural question concerns the axiomatic characterization of the inequalities $t \leq t'$ between dagger terms that hold in the models \mathbf{CPO}_m , \mathbf{CPO}_c , etc. Call an iteration category \mathcal{C} an *ordered iteration category* if it is an ordered cartesian category such that dagger is monotonic and $\perp_{B,A} \leq f$ for all $f : B \rightarrow A$, where $\perp_{B,A} = (\pi_A^{A \times B})^\dagger$. Then an inequality $t \leq t'$ holds in all cartesian categories \mathbf{CPO}_m or in any of the standard ordered models mentioned above iff it holds in all ordered iteration theories.

6 Relative Axiomatization

Several iteration categories have an additional structure, such as a partial order or an additive structure. For example, each hom-set of the cartesian category \mathbf{CL}_m or \mathbf{CL}_c is partially ordered by the pointwise ordering, and the binary supremum operation gives rise to an additive structure.

In this section, we are interested in the interaction between the iteration category structure and the additional structure. We review several relative axiomatization results showing that this interaction can often be described by a finite number of additional identities. We extend our dagger terms with the formation of terms of the form $t + t'$, where $t, t' : u \rightarrow v$. Below we will often write $t \leq t'$ as an abbreviation for the identity $t + t' = t'$, where t, t' are extended terms $u \rightarrow v$.

We start by recalling a result from [34] that provides a characterization of the identities of the models \mathbf{CL}_m or \mathbf{CL}_c equipped with binary supremum as the $+$ operation.

Theorem 12. [34] *An identity between extended terms holds in all cartesian categories \mathbf{CL}_m or \mathbf{CL}_c iff it holds in all iteration categories with an additive structure satisfying the following identities:*

$$(f + g) + h = f + (g + h), \quad f, g, h : u \rightarrow v \quad (12)$$

$$f + g = g + f, \quad f, g : u \rightarrow v \quad (13)$$

$$f + \perp_{u,v} = f, \quad f : u \rightarrow v \quad (14)$$

$$(f + g) \circ h = (f \circ h) + (g \circ h), \quad f, g : v \rightarrow w, h : u \rightarrow v \quad (15)$$

$$(\pi_{(1)}^{u^2} + \pi_{(2)}^{u^2})^\dagger = \text{id}_u \quad (16)$$

$$f^\dagger \leq (f + g)^\dagger, \quad f, g : u \times v \rightarrow u \quad (17)$$

In any model \mathcal{C} of these axioms, the $+$ operation is idempotent and defines a partial order on each hom-set: when $f, g : A \rightarrow B$, then $f \leq g$ iff $f + g = g$, so that $f + g$ is the supremum of f and g . It is clear that the operation $+$ is monotonic in both arguments, moreover, it follows by (15) that composition is monotonic in the first argument. The last axiom asserts that dagger is monotonic, and together with the other axioms implies that composition is also monotonic in the second argument.

The fixed point induction rule (11) holds in the categories \mathbf{CL}_m and \mathbf{CL}_c . Combining Theorem 12 with Theorem 10 we obtain:

Corollary 2. [34] *An identity between extended terms holds in all cartesian categories \mathbf{CL}_m or \mathbf{CL}_c iff it holds in all cartesian categories equipped with a dagger operation satisfying the fixed point (4) and parameter identities (1), the fixed point induction rule (11), and the identities (12) – (16).*

In fact, (16) may be replaced in Corollary 2 by the following identity:

$$f + f = f, \quad f : u \rightarrow v \quad (18)$$

There is a connection to formal languages, and context-free languages in particular. Suppose that Σ is a finite alphabet and consider the category whose objects are the finite sets with an A -indexed family $f = (f_a)_{a \in A}$ of languages (or just context-free languages) in $(B \cup \Sigma)^*$ as a morphism $B \rightarrow A$. Composition is defined by substitution and the identity morphism id_A is the family $(\{a\})_{a \in A}$. This category \mathbf{Lang}_Σ (or \mathbf{CF}_Σ in the context-free case) has finite products given on objects by disjoint union. Moreover, it has a $+$ operation defined by set union, and a dagger operation defined by least fixed points. Indeed, we may view a morphism $A \times B \rightarrow A$ as a ‘generalized context-free grammar’ with nonterminals in A and terminal symbols in $B \cup \Sigma$, possibly having an infinite number of rules. Then for each $a \in A$, the a -component of $f^\dagger : B \rightarrow A$ is the language generated from the nonterminal a .

Theorem 13. [39] *An identity between extended terms holds in all cartesian categories \mathbf{Lang}_Σ or \mathbf{CF}_Σ iff it holds in all iteration categories with an additive structure satisfying the identities (12) – (17).*

It is remarkable that the very same identities also characterize *simulation equivalence* [64,67] of processes, or *synchronization trees* (i.e., unfoldings of processes). For details, we refer to [34].

There are several further relative axiomatization results. We mention a few. *Bisimulation equivalence* [64,67] is weaker than simulation equivalence and can be characterized by the first five identities (12) – (16) of Theorem 12, cf. [13]. For *probabilistic* and *weighted bisimulation*, we refer to [1,40]. In order to get language equivalence, one needs to add

$$\begin{aligned} f \circ (g + h) &= (f \circ g) + (f \circ h), & f : v \rightarrow w, g, h : u \rightarrow v \\ f \circ \perp_{u,v} &= \perp_{u,w}, & f : v \rightarrow w \end{aligned}$$

Since by these identities, dagger can be replaced by a star operation and vice versa, cf. [12], this is a categorical version of Krob’s result [60] on the axiomatization of the equational theory of (regular) languages equipped with the regular operations. For extensions of Krob’s theorem to rational power series we refer to [17,46]. For implicational axiomatization of regular languages see [20,21,57–60], for rational power series [17,47], and for regular tree languages and tree series [16,31,37,38].

7 Adding Residuation

By Theorem 9, iteration categories have no finite base for their identities. The same fact holds for the identities between extended terms involving $+$ that hold in the models \mathbf{CL}_m or \mathbf{CL}_c . In this section, we consider these standard models together with $+$ and (left) residuation. Following [41], we provide a simple system of identities, involving the operations and constants of cartesian categories, dagger, $+$ and residuation, which, in addition to being sound, is complete for the set of valid identities not involving residuation. A similar program was carried out in [69] for Kleene algebras. Besides Corollary 2, the main tool of the completeness proof, borrowed from [69, 70], is that the fixed point induction rule can be transformed into an identity involving residuals.

Suppose that \mathcal{C} is a cartesian category equipped with a $+$ operation satisfying the identities (12) – (15) and (18). In particular, \mathcal{C} is an ordered cartesian category. We say that \mathcal{C} is *residuated* if \mathcal{C} is equipped with a binary operation

$$\begin{aligned} \mathcal{C}(A, C) \times \mathcal{C}(A, B) &\rightarrow \mathcal{C}(B, C) \\ (h, g) &\mapsto h \Leftarrow g \end{aligned}$$

such that $f \circ g \leq h$ iff $f \leq (h \Leftarrow g)$ for all $f : B \rightarrow C$, $g : A \rightarrow B$ and $h : A \rightarrow C$. We call $h \Leftarrow g$ the (*left*) *residual* of h by g . For example, \mathbf{CL}_m and \mathbf{CL}_c are residuated. If $h : A \rightarrow C$ and $g : A \rightarrow B$, then $h \Leftarrow g$ is the pointwise supremum of all $f : B \rightarrow C$ with $f \circ g \leq h$.

The property of being residuated can be expressed by identities.

Proposition 7. *Suppose that \mathcal{C} is a cartesian category equipped with operations $+$ and \Leftarrow satisfying (12) – (15) and (18). Then \mathcal{C} is residuated iff*

$$\begin{aligned} (h \Leftarrow g) \circ g \leq h, \quad g : u \rightarrow v, \quad h : u \rightarrow w \\ f \leq (f \circ g) \Leftarrow g, \quad f : v \rightarrow w, \quad g : u \rightarrow v \end{aligned}$$

and

$$h \Leftarrow g \leq (h + h') \Leftarrow g, \quad g : u \rightarrow v, \quad h, h' : u \rightarrow w$$

hold.

The main result of this section is:

Theorem 14. [41] *An identity between dagger terms possibly involving $+$ (but not involving \Leftarrow) holds in all cartesian categories \mathbf{CL}_m or \mathbf{CL}_c iff it holds in all cartesian categories equipped with a $+$ operation satisfying (12) – (15), (18), which are residuated and satisfy the fixed point identity (4), the parameter identity (1) and*

$$\begin{aligned} (g \Leftarrow \langle g, \text{id}_v \rangle)^\dagger \leq g, \quad g : v \rightarrow u \\ f^\dagger \leq (f + g)^\dagger, \quad f, g : u \times v \rightarrow u. \end{aligned}$$

Actually some of the identities are redundant, see [41].

References

1. Aceto, L., Ésik, Z., Ingólfssdóttir, A.: Equational axioms for probabilistic bisimilarity. In: Kirchner, H., Ringeissen, C. (eds.) *AMAST 2002*. LNCS, vol. 2422, pp. 239–254. Springer, Heidelberg (2002)
2. Arkhangelsky, K.B., Gorshkov, P.V.: Implicational axioms for the algebra of regular languages. *Dokl. Akad. Nauk USSR Ser. A* **10**, 67–69 (1967). (in Russian)
3. Arnold, A., Nivat, M.: Metric interpretations of infinite trees and semantics of non deterministic recursive programs. *Theor. Comput. Sci.* **11**, 181–205 (1980)
4. Arnold, A., Niwinski, D.: *Rudiments of μ -Calculus*. North-Holland, Amsterdam (2001)
5. Barr, M., Wells, C.: *Category theory for computing science*. Reprints Theory Appl. Categories (22), 172 (2012)
6. Bartha, M.: A finite axiomatization of flowchart schemes. *Acta Cybern.* **8**, 203–217 (1987)
7. Bekić, H.: Definable operation in general algebras, and the theory of automata and flowcharts. Technical report, IBM Vienna (1969). Reprinted in: *Programming Languages and Their Definition - Hans Bekić (1936–1982)*. LNCS, vol. 177, pp. 30–55. Springer, Heidelberg (1984)
8. Bernátsky, L., Ésik, Z.: Semantics on flowchart programs and the free Conway theories. *ITA* **32**, 35–78 (1998)
9. Bloom, S.L., Elgot, C., Wright, J.B.: Solutions of the iteration equation and extensions of the scalar iteration operation. *SIAM J. Comput.* **9**, 25–45 (1980)
10. Bloom, S.L., Ésik, Z.: Equational logic of circular data type specification. *Theor. Comput. Sci.* **63**, 303–331 (1989)
11. Bloom, S.L., Ésik, Z.: Floyd-Hoare logic in iteration theories. *J. ACM* **38**, 887–934 (1991)
12. Bloom, S.L., Ésik, Z.: *Iteration Theories*. Springer, Heidelberg (1993)
13. Bloom, S.L., Ésik, Z., Taubner, D.: Iteration theories of synchronization trees. *Inf. Comput.* **102**, 1–55 (1993)
14. Bloom, S.L., Ésik, Z.: Some quasi-varieties of iteration theories. In: Main, M.G., Melton, A.C., Mislove, M.W., Schmidt, D., Brookes, S.D. (eds.) *MFPS 1993*. LNCS, vol. 802, pp. 378–409. Springer, Heidelberg (1994)
15. Bloom, S.L., Ésik, Z.: The equational logic of fixed points (Tutorial). *Theor. Comput. Sci.* **179**, 1–60 (1997)
16. Bloom, S.L., Ésik, Z.: An extension theorem with an application to formal tree series. *J. Automata Lang. Comb.* **8**, 145–185 (2003)
17. Bloom, S.L., Ésik, Z.: Axiomatizing rational power series over natural numbers. *Inf. Comput.* **207**, 793–811 (2009)
18. Bloom, S.L., Ésik, Z.: Iteration algebras are not finitely axiomatizable. In: Gonnet, G.H., Viola, A. (eds.) *LATIN 2000*. LNCS, vol. 1776, pp. 367–376. Springer, Heidelberg (2000)
19. Bloom, S.L., Ésik, Z., Labella, A., Manes, E.G.: Iteration 2-theories. *Appl. Categorical Struct.* **9**, 173–216 (2001)
20. Boffa, M.: Une remarque sur les systèmes complets d'identités rationnelles. *ITA* **24**, 419–428 (1990)
21. Boffa, M.: Une condition impliquant toutes les identités rationnelles. *ITA* **29**, 515–518 (1995)
22. Cazanescu, V.E., Stefanescu, G.: Towards a new algebraic foundation of flowchart scheme theory. *Fund. Inform.* **13**, 171–210 (1990)

23. Cohn, P.M.: Universal algebra, 2nd edn. D. Reidel, Dordrecht (1981)
24. Davey, B.A., Priestly, H.A.: Introduction to lattices and order. Cambridge Univ. Press, Cambridge (1990)
25. De Bakker, J.W., Scott, D.: A theory of programs. Technical report, IBM Vienna (1969)
26. Elgot, C.C.: Monadic computation and iterative algebraic theories. In: Logic Colloquium 1973, Bristol. Studies in Logic and the Foundations of Mathematics, vol. 80, pp. 175–230. North-Holland, Amsterdam (1975)
27. Ésik, Z.: Identities in iterative and rational theories. *Comput. Linguist. Comput. Lang.* **14**, 183–207 (1980)
28. Ésik, Z.: Independence of the equational axioms of iteration theories. *JCSS* **36**, 66–76 (1988)
29. Ésik, Z.: A note on the axiomatization of iteration theories. *Acta Cybern.* **9**, 375–384 (1990)
30. Ésik, Z.: Completeness of park induction. *Theor. Comput. Sci.* **177**, 217–283 (1997)
31. Ésik, Z.: Axiomatizing the equational theory of regular tree languages (Extended abstract). In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 455–465. Springer, Heidelberg (1998)
32. Ésik, Z.: Group axioms for iteration. *Inf. Comput.* **148**, 131–180 (1999)
33. Ésik, Z.: Axiomatizing iteration categories. *Acta Cybern.* **14**, 65–82 (1999)
34. Ésik, Z.: Axiomatizing the least fixed point operation and binary supremum. In: Clote, P.G., Schwichtenberg, H. (eds.) CSL 2000. LNCS, vol. 1862, pp. 302–316. Springer, Heidelberg (2000)
35. Ésik, Z.: A proof of the Krohn-Rhodes decomposition theorem. *Theor. Comput. Sci.* **234**, 287–300 (2000)
36. Ésik, Z.: The power of the group axioms for iteration. *Int. J. Algebr. Comput.* **10**, 349–373 (2000)
37. Ésik, Z.: Axiomatizing the equational theory of regular tree languages. *J. Log. Algebr. Program.* **79**, 189–213 (2010)
38. Ésik, Z.: Multi-linear iterative K-semialgebras. *Electr. Notes Theor. Comput. Sci.* **276**, 159–170 (2011)
39. Ésik, Z.: A connection between concurrency and language theory. *Electr. Notes Theor. Comput. Sci.* **298**, 143–164 (2013)
40. Ésik, Z.: Axiomatizing weighted synchronization trees and weighted bisimilarity. *Theor. Comput. Sci.* **534**, 2–23 (2014)
41. Ésik, Z.: Residuated park theories. *J. Log. Comput.* **25**, 453–471 (2015)
42. Ésik, Z.: Equational axioms associated with finite automata for fixed point operations in cartesian categories. *Math. Struct. Comput. Sci.* (to appear)
43. Ésik, Z.: Equational properties of stratified least fixed points (extended abstract). In: de Paiva, V., de Queiroz, R., Moss, L.S., Leivant, D., de Oliveira, A. (eds.) WoLLIC 2015. LNCS, vol. 9160, pp. 174–188. Springer, Heidelberg (2015)
44. Ésik, Z., Bernátsky, L.: Scott induction and equational proofs. *Electr. Notes Theor. Comput. Sci.* **1**, 154–181 (1985)
45. Ésik, Z., Kuich, W.: Inductive star-semirings. *Theor. Comput. Sci.* **324**, 3–33 (2004)
46. Ésik, Z., Kuich, W.: Free iterative and iteration K-semialgebras. *Algebra Univers.* **67**, 141–162 (2012)
47. Ésik, Z., Kuich, W.: Free inductive K-semialgebras. *J. Log. Algebr. Program.* **82**, 111–122 (2013)
48. Ésik, Z., Kuich, W.: Solving fixed-point equations over complete semirings (to appear)

49. Ésik, Z., Labella, A.: Equational properties of iteration in algebraically complete categories. *Theor. Comput. Sci.* **195**, 61–89 (1998)
50. Ésik, Z., Rondogiannis, P.: A fixed point theorem for non-monotonic functions. *Theor. Comput. Sci.* **574**, 18–38 (2015)
51. Gécseg, F.: *Products of Automata*. Springer, Berlin (1986)
52. Ginzburg, A.: *Algebraic Theory of Automata*. Academic Press, New-York (1968)
53. Goguen, J.A., Thatcher, J.W., Wagner, E.G., Wright, J.B.: Initial algebra semantics and continuous algebras. *J. ACM* **24**, 68–95 (1977)
54. Hasegawa, M.: Recursion from cyclic sharing: traced monoidal categories and models of cyclic lambda calculi. In: de Groote, P., Hindley, J.R. (eds.) *TLCA 1997*. LNCS, vol. 1210, pp. 196–213. Springer, Heidelberg (1997)
55. Hyland, M., Power, J.: The category theoretic understanding of universal algebra: Lawvere theories and monads. *Electr. Notes Theor. Comput. Sci.* **172**, 437–458 (2007)
56. Joyal, A., Street, R., Verity, D.: Traced monoidal categories. *Math. Proc. Camb. Philos. Soc.* **3**, 447–468 (1996)
57. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. In: *LICS 1991*, pp. 214–225. IEEE Press (1991)
58. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. *Inf. Comput.* **110**, 366–390 (1994)
59. Krob, D.: A complete system of B-rational identities. In: Paterson, M. (ed.) *ICALP 1990*. LNCS, vol. 443, pp. 60–73. Springer, Heidelberg (1990)
60. Krob, D.: Complete systems of B-rational identities. *Theor. Comput. Sci.* **89**, 207–343 (1991)
61. Krohn, K., Rhodes, J.L.: Algebraic theory of machines, I, principles of finite semigroups and machines. *Trans. Am. Math. Soc.* **116**, 450–464 (1965)
62. Lawvere, F.W.: Functorial semantics of algebraic theories. *Proc. Nat. Acad. Sci. (USA)* **50**, 869–872 (1963)
63. Lehmann, D., Smyth, M.B.: Algebraic specification of data types: a synthetic approach. *Math. Sys. Theory* **14**, 97–139 (1981)
64. Milner, R.: *Communication and Concurrency*. Prentice-Hall, New York (1989)
65. Niwinski, D.: Equational μ -calculus. In: Skowron, A. (ed.) *Computation Theory*. LNCS, vol. 208, pp. 169–176. Springer, Heidelberg (1985)
66. Niwinski, D.: On fixed-point clones (extended abstract). In: Kott, L. (ed.) *ICALP 1986*. LNCS, vol. 226, pp. 464–473. Springer, Heidelberg (1986)
67. Park, D.M.R.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) *Theoretical Computer Science*. LNCS, vol. 104, pp. 167–183. Springer, Heidelberg (1981)
68. Plotkin, G.: *Domains*. The Pisa notes. The University of Edinburgh, Edinburgh (1983)
69. Pratt, V.: Action logic and pure induction. In: van Eijck, J. (ed.) *Logics in AI 1990*. LNCS, vol. 478, pp. 97–120. Springer, Heidelberg (1990)
70. Santocanale, L.: On the equational definition of the least prefixed point. *Theor. Comput. Sci.* **295**, 341–370 (2003)
71. Simpson, A.K., Plotkin, G.D.: Complete axioms for categorical fixed-point operators. In: *LICS 2000*, pp. 30–41. IEEE Press (2000)
72. Wagner, E.G., Bloom, S.L., Thatcher, J.W.: Why algebraic theories. In: *Algebraic Methods in Semantics*, pp. 607–634. Cambridge University Press, New York (1986)
73. Wright, J.B., Thatcher, J.W., Wagner, E.G., Goguen, J.A.: Rational algebraic theories and fixed-point solutions. In: *FOCS 1976*, pp. 147–158. IEEE Press (1976)

Reversible and Irreversible Computations of Deterministic Finite-State Devices

Martin Kutrib^(✉)

Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
kutrib@informatik.uni-giessen.de

Abstract. Finite-state devices with a read-only input tape that may be equipped with further resources as queues or pushdown stores are considered towards their ability to perform reversible computations. Some aspects of the notion of logical reversibility are addressed. We present some selected results on the decidability, uniqueness, and size of minimal reversible deterministic finite automata. The relations and properties of reversible automata that are equipped with storages are discussed, where we exemplarily stick with the storage types queue and pushdown store. In particular, the computational capacities, decidability problems, and closure properties are the main topics covered, and we draw attention to the overall picture and some of the main ideas involved.

Keywords: Reversibility · Finite state devices · Minimality · Queue and pushdown storage · Decidability · Closure properties

1 Introduction

Reversibility is a practically motivated property that has been investigated for several automata models. Computers can be seen as information processing devices which are physical realizations of such abstract models. From this viewpoint it is natural to study the fundamental physical principle of reversibility, which means in essence that every configuration has at most one unique successor configuration and at most one unique predecessor configuration. Moreover, in [29] it has been argued that only the logically irreversible operations in a computer necessarily dissipate energy by generating a corresponding amount of entropy for every irreversibly erased bit of information. This observation strongly suggests to study reversible computations without loss of information. A main question in this setting is whether or not the computation of a given automaton model can be made reversible in general.

The first investigations of reversible computations date back to the sixties of the last century when the massively parallel model of cellular automata was studied in this respect. It has been shown that the injectivity of the global transition function is equivalent to the reversibility of the automaton. It turned out that global reversibility is decidable for one-dimensional cellular automata [1], whereas the problem is undecidable for higher dimensions [16]. Nowadays it

is known from [33] that every, possibly irreversible, one-dimensional cellular automaton can always be simulated by a reversible one-dimensional cellular automaton in a constructive way.

Later, in [6] reversible sequential machines, more precisely, Turing machines have been studied. Again, a fundamental result is that every Turing machine can be made reversible or, in other words, that any recursively enumerable language can be accepted in a reversible way. Given this result, the question for the efficiency of such a simulation almost suggests itself. Let the irreversible computation take t time and s space. In [7] a first efficient reversible simulation is proposed that uses $s \cdot t^{\log(3)}$ time and $s \cdot \log(t)$ space. So, for maximal t it uses s^2 space. In [30] a different method has been shown that uses only $O(s)$ space but at the cost of exponential time. In [9] a general upper bound on the tradeoff between time and space that suffices for the reversible simulation of irreversible computations is proved. It has the exponential time simulation and the quadratic space simulation as extremes. The result shows that it is possible to achieve subexponential time and subquadratic space simultaneously.

Valuable surveys with further references to literature are, for example, [17] for cellular automata and [34], where one may find a summary of results on reversible Turing machines, reversible cellular automata, and other reversible models such as logic gates, logic circuits, or logic elements with memory, and [20] for further aspects of reversibility (see also [4, 21, 22, 25] for further investigations).

Logical reversibility has been studied also for further models such as time-bounded Turing machines [5], two-way multi-head finite automata [3, 35], one-way multi-head finite automata [24], queue automata [26], and limited automata [27].

Here we consider some aspects of reversibility in sequential devices that have a read-only input tape and may be equipped with further storage resources. The discussion is mainly restricted exemplarily to finite automata as well as to queue and pushdown automata. The notion of reversibility and its possible definitions are discussed. Then we turn to the size of reversible finite automata. It is well known that the minimal DFA accepting a given regular language is unique up to isomorphism. So the relations between minimality and reversibility are of natural interest, in particular, questions concerning the decidability, uniqueness, and the size of a minimal reversible DFA in terms of the size of the equivalent minimal DFA. Finally, the relations and properties of reversible automata that are equipped with a pushdown store or a queue are discussed, where the computational capacities, decidability problems, and closure properties are the main topics.

2 Preliminaries and the Notion of Logical Reversibility

The reader is assumed to be familiar with the basic notions of automata theory as contained, for example, in [12, 15]. In the present paper we will use the following notational conventions. An *alphabet* Σ is a non-empty finite set, its elements are called *letters* or *symbols*. We write Σ^* for the *set of all words* over the finite

alphabet Σ . The *empty word* is denoted by λ , and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The *reversal* of a word w is denoted by w^R and for the *length* of w we write $|w|$. We use \subseteq for *inclusions* and \subset for *strict inclusions*. In the following, two devices are said to be *equivalent* if they accept the same language.

The devices we are interested in are computing machines with a finite number of discrete internal states. The machines have a read-only input tape, may be equipped with further resources, and evolve in discrete time, where each computation step is driven by a *deterministic transition function*. Given a *configuration* representing the complete “global state” of a device, the transition function is used to compute the successor configuration. The transition function depends on the current internal state and on the status of further resources the machine is equipped with. It gives the successor state and maybe changes the status of the resources. In general, these devices are considered in terms of formal language recognition. However, reversibility is a property of machines and not a property of languages. So, notions as “the family of reversible regular languages” are meaningless unless the reversibility of a regular language is defined by the reversibility of a certain type of device that accepts it. For example, the deterministic finite automaton (DFA) depicted in Fig. 1 accepts the regular language a^*bb^* . Since any equivalent DFA must have a state with two incoming edges which are labeled by the same input symbol, it cannot be reversible. So, from the viewpoint of (deterministic) finite automata the language a^*bb^* is irreversible. However, in [19] it has been shown that reversible *two-way* deterministic finite automata characterize the regular languages. This implies that every regular language is accepted by some reversible two-way deterministic finite automaton and, thus, from the viewpoint of (deterministic) two-way finite automata the language a^*bb^* is reversible.

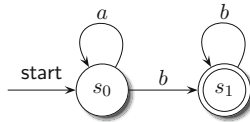


Fig. 1. An irreversible DFA accepting the language a^*bb^* , that cannot be accepted by any reversible DFA.

Basically, the definition of logical reversibility of some type of device requires that the device is deterministic and that any configuration must have at most one predecessor. But these requirements do not define reversibility sufficiently. For example, in which way is the predecessor configuration computed? May we use a universal device? Do we have to use a device of the same type? Or else a device with the same computational power? The idea to step the computation back and forth anticipates not to use a universal machine in general. For the latter question consider again the DFA of Fig. 1 that accepts the language a^*b^+ . If the predecessor configuration has to be computed by a DFA as well, the given DFA

is irreversible. Once in state s_1 , it is impossible to get back uniquely on input symbol b . However, if a DFA is equipped with an input window of size two, it can compute the predecessor state. Moreover, deterministic finite automata with window size two have the same power as DFA. So, if the predecessor configuration may be computed with lookahead two, the given DFA is reversible. Further results on gradual reversibility dependent on the size of lookaheads can be found in [2, 28].

Another question that comes up in connection with the computability of predecessor configurations concerns the set of configurations that count. Do we have to consider all possible configurations as potential predecessors? Or only configurations that are reachable from some initial configurations, that is, configurations that actually occur in computations? Consider for example the DFA in Fig. 2. It is reversible for all reachable configurations, but it is irreversible if all possible configurations count. Reversibility on reachable configurations is a wider notion than reversibility on all configurations. It turns out that this makes a big difference if *machines* are considered, but it does not make any difference if *languages* are considered (see Sect. 4).

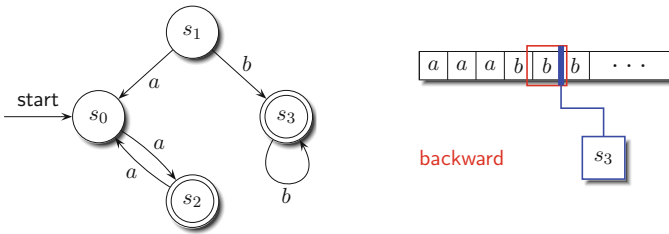


Fig. 2. [20] A DFA (left) and an unreachable configuration (right).

Unless stated otherwise, in the sequel we tacitly make the appointment that the backward steps of a computation are performed by another device of the same type and that all configurations count.

3 Size of Reversible Finite Automata

It is well known that the minimal DFA accepting a given regular language is unique up to isomorphism. So the relations between minimality and reversibility are of natural interest, in particular, questions concerning the decidability, uniqueness, and the size of a minimal reversible DFA in terms of the size of the equivalent minimal DFA.

Before we turn to the discussion of these relations, we recall some definitions. A *deterministic finite automaton* (DFA) is a system $M = \langle S, \Sigma, \delta, s_0, F \rangle$, where S is the finite set of *internal states*, Σ is the alphabet of *input symbols*, $s_0 \in S$ is the

initial state, $F \subseteq S$ is the set of *accepting states*, and $\delta: S \times \Sigma \rightarrow S$ is the *partial transition function*. Note, that here the transition function is not required to be *total*. By $\delta^-: S \times \Sigma \rightarrow 2^S$, with $\delta^-(q, a) = \{p \in S \mid \delta(p, a) = q\}$, we denote the *reverse transition function* of δ .

A DFA is *reversible* if every letter $a \in \Sigma$ induces an *injective partial mapping* from S to itself via the mapping $\delta_a: S \rightarrow S$ with $p \mapsto \delta(p, a)$. In this case, the reverse transition function δ^- can be seen as a (partial) injective function $\delta^-: S \times \Sigma \rightarrow S$.

A restricted variant of reversible deterministic finite automata has been introduced and studied in the context of algorithmic learning theory in [2]; see also [18]. The definition there requires that any reversible DFA has only one sole accepting state. Sometimes these devices are called *bideterministic* DFA. For this notion of reversibility the question for uniqueness and the size of a minimal reversible DFA is settled, as a language L is accepted by a bideterministic DFA if and only if the minimal DFA for L is reversible and has a unique final state (see [36]).

Later this concept of reversibility has been extended in [36], so that multiple accepting as well as multiple initial states are allowed. In particular, this means that reversible DFA in this sense are nondeterministic devices. However, even these devices cannot accept the regular language a^*b^* reversibly [36]. A further generalization of reversibility to quasi-reversibility, which even allows nondeterministic transitions was introduced in [32] (see also [11]). However, these quasi-reversible DFA may be exponentially more succinct than the minimal reversible DFA.

Next we turn to discuss the question for decidability, uniqueness, and the size of a minimal reversible DFA in the standard definition from above.

The example depicted in Fig. 3 answers the question whether a minimal reversible DFA is unique.



Fig. 3. Non-isomorphic minimal reversible DFA for the finite language $L = \{aa, ab, ba\}$.

Theorem 1 ([14]). *Let L be a regular language accepted by some reversible DFA. Then a minimal reversible DFA accepting L is not necessarily unique, even not up to isomorphism.*

3.1 Trade-Offs

The first exponential lower bound for the state trade-off between a minimal DFA and an equivalent minimal reversible DFA originates in [13]. By the $2n$ -fold concatenation L^{2n} of the finite language $L = \{aa, ab, ba\}$ a lower bound of $\Omega(1.001^n)$ has been derived. So, the minimal reversible finite automaton for some language can be exponentially larger than the minimal automaton. In [14] the exact number of states of a minimal reversible DFA for the language L^{2n} has been shown; it is $2^{2n+2} - 3$. Since the minimal DFA for L^{2n} has $6n + 1$ states, the blow-up in the number of states is in the order of $2^{n/3} = (\sqrt[3]{2})^n$, which is approximately 1.259^n .

However, in the same paper [14] the lower bound has been improved. A witness DFA is depicted in Fig. 4. Notice that no transitions are defined from the sole accepting state. Clearly, the DFA is minimal, but not reversible. However, since the language accepted is finite, one readily sees that it can be accepted by a reversible DFA. It is shown that the number of states of the minimal equivalent reversible DFA is $\sum_{i=1}^n F_n$, where F_n denotes the n th Fibonacci number. This is equal to $F_{n+2} - 1$. From the closed form

$$F_n = \frac{1}{\sqrt{5}} \cdot \left(\frac{1 + \sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \cdot \left(\frac{1 - \sqrt{5}}{2} \right)^n$$

and the fact that $\left(\frac{1 - \sqrt{5}}{2} \right)^n$ tends to zero, for large n , we see that the state blow-up is in the order of $\left(\frac{1 + \sqrt{5}}{2} \right)^n$, that is, approximately 1.618^n , the golden ratio Φ to the power of n .

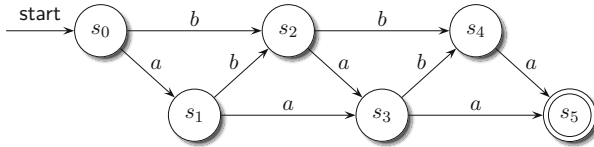


Fig. 4. [14] A minimal DFA, for $n = 6$ states, where the minimal equivalent reversible DFA needs $\sum_{i=1}^n F_i = F_{n+2} - 1$ states.

Theorem 2 ([14]). *For every n with $n \geq 3$ there is an n -state DFA over a binary input alphabet accepting a reversible language, such that any equivalent reversible DFA needs at least $\Omega(\Phi^n)$ states with $\Phi = (1 + \sqrt{5})/2$, the golden ratio.*

It is worth mentioning that the lower bound for the witness languages of Theorem 2 is for a binary alphabet. It can be increased at the cost of more symbols. For a k -ary alphabet one can derive the lower bound from the k -ary Fibonacci function $F_n = F_{n-1} + F_{n-2} + \dots + F_{n-k}$. For $k = 3$ the lower bound is of order 1.839^n and for $k = 4$ it is of order 1.927^n . For growing alphabet sizes the bound asymptotically tends to 2^{n-1} , that is, $\Omega(2^{n-1})$. This is precisely the upper bound (for arbitrary alphabet sizes).

Theorem 3 ([14]). *Let M be a minimal deterministic finite automaton with n states, that accepts a reversible language. Then a minimal reversible deterministic finite automaton for $L(M)$ has at most 2^{n-1} states.*

3.2 Decidability

Now we turn to decidability questions in connection with (minimal) reversible DFA. The first problem that comes into mind is the problem to decide whether a given DFA is reversible or, more involved, whether it accepts a *language* that is also accepted by some reversible DFA. The decision of the reversibility of DFA is almost trivial. An inspection of the transition function and the set of accepting states suffices. Moreover, this observation transfers also to languages accepted by bideterministic automata in the notion of [2], because it is sufficient to verify the reversibility of the minimal DFA for the language, which must have a unique final state (see remark above). For languages accepted by nondeterministic DFA, where the nondeterminism is limited to multiple initial states, it has been shown in [36], that there is a polynomial time algorithm for testing whether the language can be accepted by a reversible finite automaton.

For DFA the problem has been solved in [14] by proving the following structural characterization of regular languages that can be accepted by reversible DFA in terms of their minimal DFA.

Theorem 4 ([14]). *Let $M = \langle S, \Sigma, \delta, s_0, F \rangle$ be a minimal deterministic finite automaton. The language $L(M)$ can be accepted by a reversible deterministic finite automaton if and only if there do not exist useful states $p, q \in S$, a letter $a \in \Sigma$, and a word $w \in \Sigma^*$ such that $p \neq q$, $\delta(p, a) = \delta(q, a)$, and $\delta(q, aw) = q$.*

So, the characterization is based on the absence of a forbidden pattern in the (minimal) deterministic state graph. Now, checking the absence of the forbidden patterns yields an NL-complete decidability algorithm. The idea of proving NL containment is to decide in NL whether a given DFA accepts a *non-reversible* language by witnessing the forbidden pattern. Since NL is closed under complementation the containment of the reversibility problem within NL follows. For the NL hardness, the NL-complete graph reachability problem is reduced to the problem in question (with respect to deterministic logspace reductions).

Theorem 5 ([14]). *Given a DFA M , the problem to decide whether $L(M)$ is accepted by any reversible DFA is NL-complete.*

Another interesting decidability problem is to determine whether a given reversible DFA is already minimal. Again with a forbidden pattern approach, it is shown in [14] that the minimality of reversible DFA can be decided by an NL-complete algorithm.

Theorem 6 ([14]). *Given a DFA M , the problem to decide whether M is already a minimal reversible deterministic finite automaton is NL-complete.*

A further result in [14] is the *effective construction* of a minimal reversible DFA out of a given DFA that accepts a reversible language. The basic idea how to make a given DFA reversible is very intuitive: as long as there is an irreversible state, copy this state and all states reachable from it, and distribute the incoming transitions to the new copies. The absence of the forbidden pattern ensures that this procedure eventually comes to an end.

4 Queues and Pushdown Stores

This section is devoted to discuss relations and properties of reversible automata that are equipped with a pushdown store (DPDA) or a queue (DQA). Their reversible variants have been introduced and studied in [23] and [26], where only reachable configurations are relevant for reversibility. However, for pushdown automata and queue automata this makes a difference only for *machines*, that is, there are machines that are reversible on reachable but not on all configurations. It does not make a difference for *languages*, that is, for any machine that is reversible on reachable configurations there is an equivalent machine that is reversible on all configurations. So, from the perspective of languages and language classes it is safe to stick with either notion of reversibility.

Recall that a queue automaton, at each time step, may remove or keep the symbol at the front and enters a (possibly empty) symbol at the end of the queue. The transition depends on the current state, the current input symbol or λ , and the symbols currently at the front and end of the queue. Often queue automata are defined that can only see the symbol at the front of the queue. However, with an eye towards reversible computations we extend the definition as described. It is worth mentioning that the additional knowledge of the last queue symbol does not increase the computational power of queue automata. For reverse computation steps the head of the input tape is again moved to the *left*. Moreover, the roles played by the front and end of the queue are interchanged. That is, in reverse computation steps the symbols are removed from the end and added to the front of the queue. We denote the relation from one configuration to the next by \vdash .

A DQA M with transition function δ is said to be reversible (REV-DQA), if there exists a reverse transition function δ^- inducing a relation \vdash^- from one configuration to the next, so that $c' \vdash^- c$ if and only if $c \vdash c'$, for any two configurations c, c' of M (Fig. 5). See [26] for detailed definitions.

The following example is interesting insofar as the language is known not to be accepted by any reversible pushdown automaton.

Example 7 ([26]). The deterministic linear context-free language $\{a^n b^n \mid n \geq 1\}$ is accepted by the quasi realtime REV-DQA with state set $\{q_0, q_1, q_2\}$, queue alphabet $\{A_0, B_0, B_1\}$, initial state q_0 , set of accepting states $\{q_2\}$, and empty queue symbol \perp , where the transition functions δ and δ^- are as follows. Let $X \in \{A_0, B_0, B_1\}$.

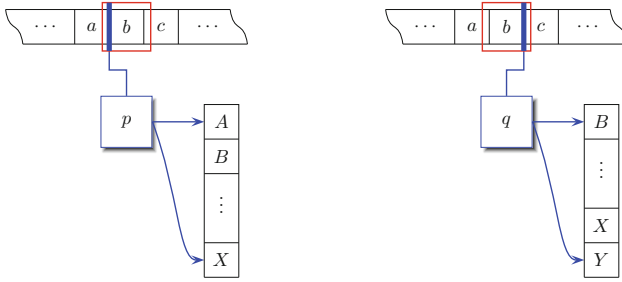


Fig. 5. Successive configurations of a REV-DQA, where $\delta(p, b, A, X) = (q, Y, \text{remove})$ (left to right) and $\delta^-(q, b, B, Y) = (p, A, \text{remove})$ (right to left).

Transition function δ	Reverse transition function δ^-
(1) $\delta(q_0, a, \perp, \perp) = (q_0, A_0, \text{keep})$	(1) $\delta^-(q_0, a, A_0, A_0) = (q_0, \lambda, \text{remove})$
(2) $\delta(q_0, a, A_0, X) = (q_0, A_0, \text{keep})$	
(3) $\delta(q_0, b, A_0, X) = (q_1, B_0, \text{remove})$	(2) $\delta^-(q_1, b, X, B_0) = (q_0, A_0, \text{remove})$
(4) $\delta(q_1, b, A_0, X) = (q_1, B_1, \text{remove})$	(3) $\delta^-(q_1, b, X, B_1) = (q_1, A_0, \text{remove})$
(5) $\delta(q_1, \lambda, B_0, B_0) = (q_2, \lambda, \text{keep})$	(4) $\delta^-(q_2, \lambda, B_0, B_0) = (q_1, \lambda, \text{keep})$
(6) $\delta(q_1, \lambda, B_0, B_1) = (q_2, \lambda, \text{keep})$	(5) $\delta^-(q_2, \lambda, B_0, B_1) = (q_1, \lambda, \text{keep})$

The main idea of the construction is to provide two flags for the enqueued symbols, namely the letter and its index. The letters characterize whether the machine is in mode *A* or mode *B*. The machine is in mode *A* while reading *a*'s from the input and is in mode *B* when reading *b*'s and checking whether the number of *a*'s and *b*'s coincide. The index describes in which state the machine was in its last step. This information is necessary for the reverse transition function.

In order to obtain a reversible automaton some transition rules have to be provided that cannot be used in any reachable configuration. For example, let the REV-DQA perform the transition $\delta(q_0, a, A_0, A_0) = (q_0, A_0, \text{keep})$. Then the reverse transition rule $\delta^-(q_0, a, A_0, A_0) = (q_0, \lambda, \text{remove})$ has to be defined. However, applying this rule to the unreachable configuration, where the queue content is $A_0B_1A_0$ gives the predecessor configuration with queue content A_0B_1 . This implies that the (forward) transition rule $\delta(q_0, a, A_0, B_1) = (q_0, A_0, \text{keep})$ has to be defined as well. ■

For the sake of completeness, recall that the transition function of a deterministic pushdown automaton maps the current state, the current input symbol or λ , and the symbol at the top of the stack to the successor state and a new (possibly empty) string at the top of the stack.

A DPDA *M* with transition function δ is said to be reversible (REV-DPDA), if there exists a reverse transition function δ^- inducing a relation \vdash^- from one configuration to the next, so that $c' \vdash^- c$ if and only if $c \vdash c'$, for any two configurations *c*, *c'* of *M*. See [23] for detailed definitions.

4.1 Computational Capacity

In order to explore the general computational capacity of reversible queue automata, the next result has been shown in [26]. It contrasts the situation for pushdown automata and complements the situation for Turing machines, where every Turing machine can be made reversible [6].

Theorem 8 ([26]). *Let M be a deterministic queue automaton. Then there exists a reversible queue automaton accepting the language $L(M)$.*

Combining this construction with the result from [37] that says that queue automata without any time restriction describe the recursively enumerable languages, we obtain that reversible queue automata and Turing machines have the same computational power.

Theorem 9 ([26]). *Every recursively enumerable language is accepted by some reversible queue automaton.*

Giving a queue automaton arbitrary time for the computation may be a little unfair compared with pushdown automata, because the former can always cycle through the queue, thus, reading the whole storage content without destroying any information. A natural possibility to overcome this advantage is studied in [10] where queue automata are considered that work in quasi realtime. Quasi realtime means that the number of consecutive λ -transitions is bounded by a constant. It is shown in [10] that quasi realtime queue automata are less powerful than queue automata without time restriction. However, for reversible queue as well as pushdown automata there is the nice correspondence that both types of automata if working in quasi realtime can be sped up to realtime.

Theorem 10 ([26]). *For every quasi realtime reversible DQA an equivalent realtime reversible DQA can effectively be constructed.*

Theorem 11 ([23]). *For every reversible DPDA an equivalent realtime reversible DPDA can effectively be constructed.*

Though both types of reversible devices have strictly less power when restricted to (quasi) realtime computations, they still can accept all regular languages. The idea is simply to simulate a given DFA, whereby the state history is remembered in the storage.

On the other hand, any language known not to be accepted in realtime by a pushdown automaton is a witness for the fact that reversible pushdown automata are strictly weaker than the general pushdown automata. This raises the natural question whether *all* realtime DPDA languages are accepted by reversible DPDA. This question has been answered negatively.

Theorem 12 ([23]). *The realtime deterministic linear context-free language $\{a^n b^n \mid n \geq 0\}$ is not accepted by any reversible DPDA.*

A similar result for queue automata has been established in [26]. In particular, a language L_{mcp} is exhibited which is accepted by some realtime queue automaton, but not by any realtime reversible queue automaton. In fact, the stronger result is obtained that any reversible queue automaton accepting L_{mcp} takes at least $\Omega\left(\frac{n^2}{\log(n)}\right)$ time steps.

Example 13 ([26]). We consider the regular language $L_{bin} = ((aa + a)(bb + b))^+$. Then the language

$$L_{mcp} = \{ p\$w_1\$w_1\$w_2\$w_2\$ \cdots \$w_n\$w_n \mid p \in L_{bin}, n \geq 0, w_i \in \{a, b\}^* \}$$

is accepted by a realtime DQA. Informally, a DQA works as follows. The prefix up to the first $\$$ can be tested without using the queue, because it belongs to a regular language. Then the first copy of each w_i is stored in the queue and subsequently compared and removed from the queue while reading the second copy. Whenever the second copy matches the first copy, the automaton enters an accepting state. Whenever a mismatch is detected, a non-accepting state is entered so that the input is rejected. ■

By using Kolmogorov complexity and incompressibility arguments, the lower bound mentioned above has been shown.

Theorem 14 ([26]). *Any reversible DQA accepting L_{mcp} has a time complexity of $\Omega\left(\frac{n^2}{\log(n)}\right)$.*

This lower bound result raises the question for the costs of simulating any realtime DQA, not necessarily reversible, by an equivalent reversible DQA. It turned out that quadratic time is sufficient for such simulations.

Theorem 15 ([26]). *Every realtime DQA can be simulated by a reversible DQA that needs at most quadratic time.*

This upper bound shows that quadratic time is the trade-off for making realtime DQA reversible. On the other hand, the language L_{mcp} provides a lower bound, which shows that there are cases where a quadratic time trade-off is almost reached. Moreover, we have derived that for both types of automata the reversible variant is strictly weaker than the realtime general variant.

We conclude the subsection by an incomparability result showing that the language accepting capabilities of reversible (quasi) realtime DQA are different from those of reversible pushdown automata. The context-free language $\{ w\#w^R \mid w \in \{a, b\}^+ \}$ is accepted by a reversible pushdown automaton [23], but not by any even irreversible quasi realtime queue automaton [8, 31]. On the other hand, it is not hard to see that the non-context-free language $\{ a^n b^n c^n \mid n \geq 1 \}$ is accepted by some realtime reversible DQA.

Theorem 16 ([26]). *The families of languages accepted by realtime reversible DQA and by reversible pushdown automata are incomparable.*

4.2 Decidability and Closure Properties

Let us now turn to decidability aspects of reversible pushdown and queue automata. Problems which are decidable for DPDA are decidable for reversible DPDA as well.

Corollary 17. *Finiteness, infiniteness, universality, equivalence, and regularity are decidable for reversible DPDA.*

On the other hand, inclusion is known to be undecidable for DPDA. By reduction of the Post's correspondence problem it has been shown that inclusion is undecidable for reversible DPDA, too.

Theorem 18 ([23]). *Inclusion is undecidable for reversible DPDA.*

The situation for reversible queue automata is in considerable contrast to the situation for reversible pushdown automata. By reduction of the emptiness problem for deterministic linearly space bounded one-tape, one-head Turing machines, so-called linear bounded automata, it has been shown that the emptiness problem for realtime reversible DQA is not even semidecidable. From this result it is derived that all the commonly studied decidability questions are non-semidecidable, too.

Theorem 19 ([26]). *Emptiness, finiteness, infiniteness, universality, inclusion, equivalence, regularity, and context-freeness are not semidecidable for realtime reversible DQA.*

These results bring us to the problem whether reversibility itself is decidable. Here again, we have to distinguish between the reversibility of a machine and the reversibility of the language accepted by a machine. Let us first shortly consider the languages. The following theorem contrasts the situation for finite automata, where the problem is decidable (see above).

Theorem 20 ([23]). *It is undecidable whether the language accepted by a non-deterministic pushdown automaton can be accepted by a reversible DPDA.*

The same problem for deterministic pushdown automata is open.

Next, we consider the question of whether the reversibility of a machine in question can be decided. Now we have to distinguish between the reversibility on all or only on reachable configurations. This makes a big difference. While the question of reversibility on *all configurations* is decidable just by inspection of the transition function, the question of reversibility on reachable configurations becomes more involved. In particular, we obtain a difference between queue and pushdown automata.

Theorem 21 ([26]). *Reversibility on reachable configurations is not semidecidable for realtime DQA.*

However, we have the decidability for pushdown automata. The size of a pushdown automaton is the length of its representation.

Theorem 22 ([23]). *Let M be a deterministic pushdown automaton of size n . Then it is decidable in time $O(n^4)$, whether M is reversible on reachable configurations. Moreover, the decision problem is P -complete.*

Given a nondeterministic pushdown automaton, by inspecting the transition function one can decide whether or not it is a DPDA. If the answer is yes, then it can be decided whether it is reversible on reachable configurations by the previous theorem. If it is not a DPDA, then it cannot be a reversible DPDA. Therefore, the previous result transfers to nondeterministic devices.

Corollary 23 ([23]). *Let M be a nondeterministic pushdown automaton of size n . Then it is decidable in time $O(n^4)$, whether M is reversible on reachable configurations. Moreover, the decision problem is P -complete.*

Finally, we consider closure properties of the families in question. It turns out that the families of languages accepted by realtime reversible queue automata and reversible pushdown automata have similar closure properties. For example, they are closed under complementation and inverse homomorphism, but are not closed under union, intersection, intersection with regular languages, concatenation, reversal, and homomorphism. The closure properties of the language families discussed are summarized in Table 1. The closure properties for general (realtime) DQA may be found in [10].

Table 1. Closure properties of language families discussed. REG denotes the family of regular languages.

Language class	\cup	\bullet	R	h_λ	h^{-1}	\cap_{REG}	\cup_{REG}	\cap	\sim
REG	+	+	+	+	+	+	+	+	+
realtime REV-DQA	-	-	-	-	+	-	-	-	+
realtime DQA	-	-	-	-	+	+	+	-	+
REV-DPDA	-	-	-	-	+	-	-	-	+
DPDA	-	-	-	-	+	+	+	-	+

References

1. Amoroso, S., Patt, Y.N.: Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures. *J. Comput. Syst. Sci.* **6**, 448–464 (1972)
2. Angluin, D.: Inference of reversible languages. *J. ACM* **29**, 741–765 (1982)
3. Axelsen, H.B.: Reversible multi-head finite automata characterize reversible logarithmic space. In: Dediu, A.-H., Martín-Vide, C. (eds.) *LATA 2012*. LNCS, vol. 7183, pp. 95–105. Springer, Heidelberg (2012)
4. Axelsen, H.B., Glück, R.: A simple and efficient universal reversible turing machine. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) *LATA 2011*. LNCS, vol. 6638, pp. 117–128. Springer, Heidelberg (2011)

5. Axelsen, H.B., Jakobi, S., Kutrib, M., Malcher, A.: A hierarchy of fast reversible turing machines. In: Krivine, J., Stefani, J.B. (eds.) *Reversible Computation (RC 2015)*. LNCS, vol. 9138, pp. 29–44. Springer, Heidelberg (2015)
6. Bennett, C.H.: Logical reversibility of computation. *IBM J. Res. Dev.* **17**, 525–532 (1973)
7. Bennett, C.H.: Time/space trade-offs for reversible computation. *SIAM J. Comput.* **18**, 766–776 (1989)
8. Brandenburg, F.J.: Intersections of some families of languages. In: Kott, L. (ed.) *International Colloquium on Automata, Languages and Programming (ICALP 1986)*. LNCS, vol. 226, pp. 60–68. Springer, Heidelberg (1986)
9. Buhrman, H., Tromp, J., Vítányi, P.M.B.: Time and space bounds for reversible simulation. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) *ICALP 2001*. LNCS, vol. 2076, pp. 1017–1027. Springer, Heidelberg (2001)
10. Cherubini, A., Citrini, C., Crespi-Reghizzi, S., Mandrioli, D.: QRT FIFO automata, breadth-first grammars and their relations. *Theoret. Comput. Sci.* **85**, 171–203 (1991)
11. García, P., de Parga, M.V., López, D.: On the efficient construction of quasi-reversible automata for reversible languages. *Inform. Process. Lett.* **107**, 13–17 (2008)
12. Harrison, M.A.: *Introduction to Formal Language Theory*. Addison-Wesley, Reading (1978)
13. Héam, P.C.: A lower bound for reversible automata. *RAIRO Inform. Théor.* **34**, 331–341 (2000)
14. Holzer, M., Jakobi, S., Kutrib, M.: Minimal reversible deterministic finite automata. In: Potapov, I. (ed.) *Developments in Language Theory (DLT 2015)*. LNCS, Springer (to appear 2015)
15. Hopcroft, J.E., Ullman, J.D.: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Boston (1979)
16. Kari, J.: Reversibility and surjectivity problems of cellular automata. *J. Comput. Syst. Sci.* **48**, 149–182 (1994)
17. Kari, J.: Reversible cellular automata. In: De Felice, C., Restivo, A. (eds.) *DLT 2005*. LNCS, vol. 3572, pp. 57–68. Springer, Heidelberg (2005)
18. Kobayashi, S., Yokomori, T.: Learning approximately regular languages with reversible languages. *Theoret. Comput. Sci.* **174**, 251–257 (1997)
19. Kondacs, A., Watrous, J.: On the power of quantum finite state automata. In: *Foundations of Computer Science (FOCS 1997)*, pp. 66–75. IEEE Computer Society (1997)
20. Kutrib, M.: Aspects of reversibility for classical automata. In: Calude, C.S., Freivalds, R., Kazuo, I. (eds.) *Gruska Festschrift*. LNCS, vol. 8808, pp. 83–98. Springer, Heidelberg (2014)
21. Kutrib, M., Malcher, A.: Fast reversible language recognition using cellular automata. *Inform. Comput.* **206**, 1142–1151 (2008)
22. Kutrib, M., Malcher, A.: Real-time reversible iterative arrays. *Theoret. Comput. Sci.* **411**, 812–822 (2010)
23. Kutrib, M., Malcher, A.: Reversible pushdown automata. *J. Comput. Syst. Sci.* **78**, 1814–1827 (2012)
24. Kutrib, M., Malcher, A.: One-way reversible multi-head finite automata. In: Glück, R., Yokoyama, T. (eds.) *RC 2012*. LNCS, vol. 7581, pp. 14–28. Springer, Heidelberg (2013)

25. Kutrib, M., Malcher, A., Wendlandt, M.: Real-time reversible one-way cellular automata. In: Isokawa, T., Imai, K., Matsui, N., Peper, F., Umeo, H. (eds.) AUTOMATA 2014. LNCS, vol. 8996, pp. 56–69. Springer, Heidelberg (2015)
26. Kutrib, M., Malcher, A., Wendlandt, M.: Reversible queue automata. In: Bensch, S., Freund, R., Otto, F. (eds.) Non-Classical Models of Automata and Applications (NCMA 2014), vol. 304, pp. 163–178. Austrian Computer Society, Vienna (2014). www.books@ocg.at
27. Kutrib, M., Wendlandt, M.: Reversible limited automata. In: Machines, Computations, and Universality (MCU 2015). LNCS, Springer (to appear, 2015)
28. Kutrib, M., Worsch, T.: Degrees of reversibility for DFA and DPDA. In: Yamashita, S., Minato, S. (eds.) RC 2014. LNCS, vol. 8507, pp. 40–53. Springer, Heidelberg (2014)
29. Landauer, R.: Irreversibility and heat generation in the computing process. IBM J. Res. Dev. **5**, 183–191 (1961)
30. Lange, K.J., McKenzie, P., Tapp, A.: Reversible space equals deterministic space. J. Comput. Syst. Sci. **60**, 354–367 (2000)
31. Li, M., Longpré, L., Vitányi, P.M.B.: The power of the queue. SIAM J. Comput. **21**, 697–712 (1992)
32. Lombardy, S.: On the construction of reversible automata for reversible languages. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, p. 170. Springer, Heidelberg (2002)
33. Morita, K.: Reversible simulation of one-dimensional irreversible cellular automata. Theoret. Comput. Sci. **148**, 157–163 (1995)
34. Morita, K.: Reversible computing and cellular automata - a survey. Theoret. Comput. Sci. **395**, 101–131 (2008)
35. Morita, K.: Two-way reversible multi-head finite automata. Fund. Inform. **110**, 241–254 (2011)
36. Pin, J.E.: On reversible automata. In: Simon, I. (ed.) LATIN 1992. LNCS, vol. 583, pp. 401–416. Springer, Heidelberg (1992)
37. Vollmar, R.: Über einen Automaten mit Pufferspeicherung. Computing **5**, 57–70 (1970)

Robust Inference and Local Algorithms

Yishay Mansour^(✉)

Microsoft Research, Hertzelia and Tel-Aviv University, Hertzelia, Israel
mansour.yishay@gmail.com

Abstract. We introduce a new feature to inference and learning which we call *robustness*. By robustness we intuitively model the case that the observation of the learner might be corrupted. We survey a new and novel approach to model such possible corruption as a zero-sum game between an adversary that selects the corruption and a learner that predicts the correct label. The corruption of the observations is done in a worst-case setting, by an adversary, where the main restriction is that the adversary is limited to use one of a fixed known class of modification functions. The main focus in this line of research is on *efficient* algorithms both for the inference setting and for the learning setting. In order to be efficient in the dimension of the domain, one cannot hope to inspect all the possible inputs. For this, we have to invoke local computation algorithms, that inspect only a logarithmic fraction of the domain per query.

1 Introduction

1.1 Motivating Scenarios

We advocate for a notion of *robustness* which combines a worst-case analysis with probabilistic approaches of inference and learning. We start with a few motivating scenarios, that highlight the need to address robustness in both inference and learning settings.

Spam filter: Consider a spam filter that uses multiple detectors to identify whether a given e-mail is spam. The classical inference problem would be to compute the probability that the e-mail is spam given the detectors' output (taking in to account their success probabilities). However, spammers are far from being passive. They continuously adapt to the spam filters updates, trying to bypass them. In the short time horizon we can expect spammers to "fool" only a few detectors, while in the long time horizon they might be able to completely fool the spam filter. It is all about a game, a game between the spammers and the detectors where spammers adjust to detectors and try to fool them. By robustness, we emphasize our desire to classify correctly whether an e-mail is spam, even when the output of a few detectors is corrupted adversarially (by the spammer).

This research was supported in part by The Israeli Centers of Research Excellence (I-CORE) program, (Center No. 4/11), by a grant from the Israel Science Foundation (ISF), by a grant from United States-Israel Binational Science Foundation (BSF).

Hardware failure: As a second motivating domain is hardware failures. Consider a network setting where multiple detectors are used to identify failures. Each detector has a different success probability of detecting correctly that a failure occurred. A classical inference question is to determine the probability that a failure occurred, given the detectors signals. The detectors themselves are additional hardware that is added to the network, and like any hardware, they may fail. This raises the question: what do you do when the detectors themselves are faulty? In general, a Bayesian approach to modeling would add a prior over any uncertainty that might occur. In this case it implies to give a probability distribution over the various detectors to fail and their output once they fail. While estimating the failure probability is plausible, it is highly optimistic to assume that we have an accurate model of the device output once it fails. For a concrete example consider a device that measures the bit error rate on a link. Detecting that the device is not responding is rather easy, but what happens in the device claims a 1% bit error rate on the link when actually there are no errors. What probability should we give this even? This is the kind of issues that the Bayesian approach has to address. In contrast, our robustness modeling takes a worst case approach. For example, we would like to guarantee that *any* single detector failure would be overcome by the system. This *worst case* requirement leads us to an adversarial modeling of the detectors' failure and deriving an optimal robust prediction given our observations is aimed at overcoming worse case failures.

Another hardware failure related domain is sensors. Inferences based on sensors' inputs have become a central aspect of wearable computing. One of the main challenges is that sensors are noisy, and information from a variety of different sources should be integrated in order to verify particular facts. In the case of bio-sensors for example, sensors of the same kind may measure a particular feature, such as blood pressure, in different parts of the body, and their readings should be integrated. At first sight this setting fits squarely into the context of probabilistic inference. However, in many such settings some inputs (sensors) might be corrupted in an unforeseeable manner (in addition to noise). This unforeseeable corruption leads us naturally to a worst case modeling, where we would like to provide the best inference guarantees assuming a few of the sensors might "maliciously" (namely, worst case) malfunction.

Strategic behavior: Consider the following classification setting. Inputs, described by attributes, are to be classified as having one of two labels. For example, an input might represent an employee, the attributes might represent her performance under various measures, and the classification function maps the values of the attributes to a binary decision, such as "deserves a bonus" or not. Suppose now that an adversary (a manager in the company? a trade union?) can manipulate the attributes of inputs in some limited way. For example, the adversary might round up or down the number of work hours that the employee reports, or might over- or under-emphasize the employee's contribution to a completed project. We refer to such manipulations as corruption of the input. As a consequence of this corruption, some employees that deserve a bonus (according to the intended classification) might not get one, and vice versa. Robustness has

the goal to design a classification algorithm that is immune to such corruption of inputs. Thus, even if the algorithm observes only the corrupted input, in most cases, the label that it predicts should match the label of the original (unknown) uncorrupted input.

1.2 Model: Overview

Each instance (x, y) is composed from a set of attributes $x \in \mathcal{X}$ and a label $y \in \{0, 1\}$, and there is a joint distribution D over (x, y) . The classical inference problem, is given x to predict y , while the classical learning problem is given a sample $S = \{(x_i, y_i)\}$ select a hypothesis h .

For our *robustness* modeling we add another layer which maps the x (uncorrupted input) to z (corrupted input). The learner now observes z and needs to predict y . The main issue is how the corrupted inputs (z) are generated from the uncorrupted ones (x).

The main feature of our model is that the corruption is done by an adversary, which can map x to z using one of m modification rules ρ , i.e., $\rho(x) = z$. The main limitation of the adversary is that the set of modification rules is fixed in advance, and robustness is done with respect to it.

The model is now of a zero-sum game between the learner and the adversary, where the learner selects a policy that maps observed signals (z) to predictions (distributions over y) and the adversary selects a modification rule. The value of the game is considered as the *optimal error rate*.

For the adversary we distinguish between two settings. In the *static* setting the adversary selects the modification rule ρ *before* the uncorrupted inputs (x) is realized. In the *dynamic* setting the adversary selects the modification rule ρ *after* the uncorrupted inputs (x) is realized and based on it. Clearly, in the dynamic setting the adversary has more power, which should intuitively translate also to a higher optimal error rate.

For the learner we distinguish between two settings. In the *learning* setting, the learner has a hypothesis class H and needs to select a hypothesis from that class based on an observed sample. In the *inference* setting, the learner can use an arbitrary prediction function, but has the additional power of querying the joint distribution D on various inputs. (Intuitively, this can be viewed as an oracle to solve the inference problem without corruption.)

1.3 Results: Highlights

The main results in [6, 10] show that one can achieve efficient algorithms for the inference and learning problem.

For the static inference setting, there is a randomized algorithm, that given as an input the corrupted instance z output a prediction $h(z)$. The algorithm receives as an input a parameter $\epsilon > 0$ and guarantees that the expected error rate is at most $error^* + \epsilon$, where $error^*$ is the optimal error rate, i.e., the value of the zero-sum game between the adversary and the learner. The running time

of the learner is polynomial in m , the number of modification rules, and n , the dimension of \mathcal{X} . The above is the main result of [10].

For the dynamic inference setting we have a similar result from [6]. The main difference is that the running time of the algorithm is not polynomial in m (but is polynomial in n). This implies that we have efficient algorithms only for a constant number of modification rules, i.e., $m = O(1)$.

For the learning setting is presented and address in [6]. In this setting we have a hypothesis class \mathcal{H} . We assume that we are given a sample of uncorrupted examples, and the goal is to learn a hypothesis which will do well on possibly corrupted inputs. Given a sample, we can find a mixture of hypotheses from \mathcal{H} which will approximate well the best hypothesis for the given sample. Assuming a large enough sample, we can show that the difference between the error rate on the sample and on the distribution is negligible, and hence we have generalization. The results for the learning setting appear in [6].

1.4 Algorithmic Techniques

In order to show the efficient inference results [6, 10] one need to invoke local computation algorithms.

For the static setting, the work of [10] uses a Linear Programming approach. It shows that the exact optimal policy can be written as a solution of an exponential size linear program. In order to derive an efficient algorithm, the linear program special structure is exploited, and a careful sampling of the constraints guarantees that the solution is a good approximation of the desired optimal policy.

For the dynamic setting, the work of [6] shows that the optimal adversary policy can be derived from a maximum matching on some bipartite graph, and the optimal learner policy can be derived from a minimum vertex cover on the same graph. Local computation algorithm for matching and vertex cover are used to efficiently implement a near optimal policy for the learner and the adversary.

2 Related Work

A general direction of related work is *inference* in Bayesian setting (see, e.g., [9, 14]). Given a partial observation of the realization of some of the variables, the goal is to compute the probability distribution of other variables.

There is a vast literature in statistics and machine learning regarding various noise models. At large, most noise models assume a random process that generates the noise. In computational learning theory the popular noise models include: random classification noise [2], and malicious noise [8, 16]. In the malicious noise model, the adversary gets to arbitrarily corrupt some small fraction of the examples; in contrast, in our model the adversary can always corrupt every example, but only in a limited way.

Another vast literature which is remotely related is that of robust optimization and stability, where the main task is to understand how minor changes in

the input can affect the outcome of a (specific) algorithm (see, [4]). We differ in two important ways from that literature. First, we do not assume any metric for our corruptions, while in large part the robustness there is measured as a function of the norm of the modification. Second, that literature, to a large extent, discusses the stability and robustness of specific algorithms, while our starting point is trying to derive algorithms immune to such corruption.

Our work establishes a solid connection between local computation algorithms (LCAs) and efficient learning. Local Computation Algorithms (LCA) were introduced in [15], and LCAs for combinatorial problems such as maximal independent set and hypergraph 2-coloring were derived in [1, 11, 15]. We build on the work of [12] that gave an LCA which finds a $(1 - \epsilon)$ -approximation to the maximum matching to derive our near optimal inference. (An alternative deterministic algorithm is given by [5].)

This line of work has also connections to Boolean functions analysis, where the uncorrupted input can be viewed as an input to the Boolean function. That literature is mainly concerned with product distributions, and even more specifically uniform distribution. Those works are interested in the study of *balanced* function, which minimizes the maximal *influence* over all the variables [3, 7]. While minimizing the *average noise sensitivity* and *maximum influence* is commonly used [13]. In some sense we are essentially minimizing the *minimum noise sensitivity*.

3 Model

There is a finite domain \mathcal{X} from which uncorrupted inputs are generated, and a finite domain \mathcal{Z} (possibly the same as \mathcal{X}) which is the domain of corrupted inputs. An arbitrary unknown target function $f : \mathcal{X} \rightarrow \{0, 1\}$ maps each uncorrupted input $x \in \mathcal{X}$ to its classification $f(x) = y \in \{0, 1\}$. There is a distribution D over the uncorrupted inputs \mathcal{X} .

The adversary maps an uncorrupted input $x \in \mathcal{X}$ to a corrupted input $z \in \mathcal{Z}$ (thus implicitly corrupting the input). There is a set R of m modification rules, where $\rho_i : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$.

For the inference setting, the learner has the ability to query the joint distribution D on inputs (x, y) and also sample inputs (x, y) from the distribution D . We will have two variants of the model, the static setting and the dynamic setting. In the *static* setting the adversary selects the modification rule $\rho_i \in R$ independent of the realized input x . In the dynamic setting, the adversary selection may depend on the realized (x, y) .¹ In the dynamic setting we will also assume that each corrupted input z can be generated by at most m distinct (x, y) inputs, i.e., for any $z \in \mathcal{Z}$ we have $|\{(x, y) : \exists i \rho_i(x, y) = z\}| \leq m$.²

¹ One can clearly map the dynamic setting to the static setting, by encoding the adversary action in the modification rules, but this will imply that the number of modification rules would be $m^{|\mathcal{X}|}$.

² This also implicitly implies that in the dynamic setting $|\mathcal{X}|/m \leq |\mathcal{Z}| \leq m|\mathcal{X}|$.

In the learning setting, the main difference is that the learner is limited to a given hypothesis class \mathcal{H} (which is used also for as a benchmark). Unlike the inference setting, here the learner cannot query the distribution D and is limited to a sample from D .

There is a loss function ℓ that measures the loss of the learner's prediction. The optimal loss rate in the static model is

$$\text{error}^* = \min_h \max_{\rho_i} E_{(x,y) \sim D}[\ell(h(\rho_i(x, y)), y)],$$

while for the dynamic model it is

$$\text{error}^* = \min_h E_{(x,y) \sim D}[\max_{\rho_i} \ell(h(\rho_i(x, y)), y)].$$

The loss function that we use is the absolute loss, i.e., $\ell(a, y) = |a - y|$, which for $y \in \{0, 1\}$ is equivalent to the prediction error.

4 Main Results

In this section we outline the main results from [6, 10]. The first result is an efficient algorithm for inference in the static model.

Theorem 1 ([10, Theorem 4.1]). *In the static inference setting, there exists an ϵ -optimal randomized policy $\hat{\pi}$, that given an observed signal $z \in \mathcal{Z}$ computes $\hat{\pi}(z)$ in time polynomial in $(n, m, 1/\epsilon)$.*

For the dynamic inference setting, we define a *interference graph*. Let $X_b = \{x : f(x) = b\}$. The collision graph is a bipartite graph, where one side has X_0 and the other is X_1 . There is an edge between $x_0 \in X_0$ and $x_1 \in X_1$ if the adversary can map both of them to the same corrupted input, i.e., there is a $z \in \mathcal{Z}$ and a modification rule ρ_i such that $\rho_i(x_0) = \rho_i(x_1) = z$. A weighted inference graph has each node x weighted by its probability $D(x)$.

The following theorem characterizes the optimal policies for the adversary and the learners as graph properties of the inference graph.

Theorem 2 ([6, Theorem 5]). *The adversary can guarantee that the expected fraction of errors (relative to the distribution D) is at least the weight of the maximum vertex-weighted fractional matching in the weighted interference graph. The learner can guarantee that the expected fraction of errors (relative to the distribution D) is at most the weight of the minimum weighted vertex cover in the weighted interference graph.*

We now need to exhibit local computation algorithms for computing a matching and a vertex cover, in order to define efficiently the optimal adversary and learner policies. For unweighted matching the following establishes a local computation algorithm.

Theorem 3 ([12, Theorem 3]). *There exists a local computation algorithm with the following properties. Given $\epsilon > 0$, and a graph $G = (V, E)$ of bounded degree d , the algorithm replies whether any given edge of G is in some matching in time $O(\log^4 |V|)$, using memory $O(\log^3 |V|)$, and with failure probability at most $1/|V|$. Furthermore, its responses to such edge queries are all consistent with some maximal matching of G which is a $(1 - \epsilon)$ -approximation to the maximum matching. (The big-Oh notation used here and latter hides constants which may depend on d and $1/\epsilon$.)*

It is well known that for bipartite graph, the maximum matching equals the minimum vertex cover. In addition, there are simple transformations that given a maximum matching produce a minimum vertex cover. However, we have only an approximate maximum matching, and the know transformations fail in that case. For this the following theorem was established.

Theorem 4 ([6, Theorem 12]). *Given a fractional matching M with no augmenting paths of length $2\ell + 1$ or less, there is a randomized algorithm such that expected weight of the output S of algorithm is at most $(1 + \frac{1}{\ell})y(M)$, where $y(M)$ is the weight M .*

Combining the vertex cover local algorithm with the prediction strategy, establishes the following theorem.

Theorem 5 ([6, Theorem 14]). *In the dynamic inference setting, there exists an ϵ -optimal randomized policy $\hat{\pi}$, that given an observed signal $z \in \mathcal{Z}$ computes $\hat{\pi}(z)$ in time polynomial in $(n, 1/\epsilon)$.*

For the learning setting [6] the following establishes the learning.

Theorem 6 ([6, Theorem 4]). *Given a hypothesis class \mathcal{H} and an ERM oracle for \mathcal{H} , there is an algorithm that, with probability $1 - \delta$, computes an ϵ -optimal learner hypothesis. The running time of the algorithm is polynomial in $1/\epsilon$, $\log 1/\delta$, m and $\log |\mathcal{H}|$.*

References

1. Alon, N., Rubinfeld, R., Vardi, S., Xie, N.: Space-efficient local computation algorithms. In: SODA, pp. 1132–1139 (2012)
2. Angluin, D., Laird, P.: Learning from noisy examples. *Mach. Learn.* **2**(4), 343–370 (1988)
3. Ben-Or, M., Linial, N.: Collective coin flipping, robust voting schemes and minima of banzhaf values. In: FOCS, pp. 408–416 (1985)
4. Ben-Tal, A., El Ghaoui, L., Nemirovski, A.S.: *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, Princeton (2009)
5. Even, G., Medina, M., Ron, D.: Best of two local models: local centralized and local distributed algorithms. In: CoRR, abs/1402.3796 (2014)
6. Feige, U., Mansour, Y., Schapire, R.: Learning and inference in the presence of corrupted inputs. In: COLT (2015)

7. Kahn, J., Kalai, G., Linial, N.: The influence of variables on boolean functions. In: FOCS, pp. 68–80 (1988)
8. Kearns, M.J., Li, M.: Learning in the presence of malicious errors. *SIAM J. Comput.* **22**(4), 807–837 (1993)
9. MacKay, D.J.C.: *Information Theory. Inference and Learning Algorithms*. Cambridge University Press, New York (2002)
10. Mansour, Y., Rubinfeld, R., Tennenholtz, M.: Robust probabilistic inference. In: *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4–6, 2015*, pp. 449–460 (2015)
11. Mansour, Y., Rubinfeld, R., Vardi, S., Xie, N.: Converting online algorithms to local computation algorithms. *ICALP* **1**, 653–664 (2012)
12. Mansour, Y., Vardi, S.: A local computation approximation scheme to maximum matching. In: *APPROX-RANDOM*, pp. 260–273 (2013)
13. Mossel, E., O’Donnell, R., Oleszkiewicz, K.: Noise stability of functions with low influences invariance and optimality. In: FOCS, pp. 21–30 (2005)
14. Pearl, J.: *Causality: Models, Reasoning, and Inference*. Cambridge University Press, New York (2000)
15. Rubinfeld, R., Tamir, G., Vardi, S., Xie, N.: Fast local computation algorithms. In: *ICS*, pp. 223–238 (2011)
16. Valiant, L.G.: Learning disjunction of conjunctions. In: *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI 1985*, pp. 560–566 (1985)

Logic, Semantics, Automata and Theory of Programming

Uniform Generation in Trace Monoids

Samy Abbes¹(✉) and Jean Mairesse²

¹ Université Paris Diderot/PPS CNRS UMR 7126
and IRISA/INRIA CNRS UMR 6074, Paris, France
samy.abbes@univ-paris-diderot.fr

² UPMC/LIP6 CNRS UMR 7606, Paris, France
jean.mairesse@lip6.fr

Abstract. We consider the problem of random uniform generation of traces (the elements of a free partially commutative monoid) in light of the uniform measure on the boundary at infinity of the associated monoid. We obtain a product decomposition of the uniform measure at infinity if the trace monoid has several irreducible components—a case where other notions such as Parry measures, are not defined. Random generation algorithms are then examined.

Keywords: Trace monoid · Uniform generation · Möbius polynomial

1 Introduction

Uniform generation of finite-size combinatorial objects consists in the design of a randomized algorithm that takes an integer k as input, and returns an object of size k , such that each object of size k has equal probability to be produced. This problem has been considered for many classes of objects from computer science or discrete mathematics: words, trees, graphs are examples. Several general approaches exist: recursive methods [11], the Markov chain Monte-Carlo method with coupling from the past [12], or the Boltzmann sampler [10]. Other recent approaches share a common guideline, namely first considering a notion of *uniform measure on infinite objects* in order to gain, afterwards, information on the uniform distributions on finite objects. The theory of random planar graphs is an example of application of this idea. In this paper, we investigate the uniform generation of traces (elements of a trace monoid) and we base our approach on the notion of *uniform measure on infinite traces*.

Given an independence pair (A, I) , where I is an irreflexive and symmetric relation on the finite alphabet A , the associated trace monoid $\mathcal{M} = \mathcal{M}(A, I)$ contains all congruence classes of the free monoid A^* , modulo equivalences of the form $ab = ba$ for all $(a, b) \in I$, see [6, 8]. Elements of \mathcal{M} are called *traces*. Trace monoids are ubiquitous in Combinatorics, see [18]. They are also one of the most basic models of concurrency under a partial order semantics [9]. Uniform generation of traces is thus a fundamental question with possible applications in probabilistic model checking of concurrent systems. Since our concern is with

partial order semantics, it differs from the sequential approach which targets uniform generation of linear executions in models of concurrency [5].

Consider a trace monoid \mathcal{M} , and, for each integer $k \geq 0$, the finite set $\mathcal{M}_k = \{x \in \mathcal{M} : |x| = k\}$. Let $\nu_{\mathcal{M}_k}$ be the uniform distribution over \mathcal{M}_k . A crucial observation is that the probability measures $(\nu_{\mathcal{M}_k})_{k \in \mathbb{N}}$ are *not* consistent. Consequently, the uniform measures $\nu_{\mathcal{M}_k}$ cannot be reached by a recursive sampling of the form $x_1 \dots x_k \in \mathcal{M}_k$, with the x_i 's being sampled independently and according to some common distribution over A .

To overcome the difficulty, several steps are necessary. First, we consider the *uniform measure at infinity* for \mathcal{M} , a notion introduced in [2] for irreducible trace monoids, and extended here to the general case. Second, we prove a realization result for the uniform measure at infinity by means of a Markov chain on a combinatorial sub-shift. Last, we apply the results to the uniform sampling of finite traces. None of the three steps is straightforward. Besides standard uniform sampling, it turns out that evaluating the uniform average cost or reward associated with traces can be done in an efficient way.

An original feature of our approach is to define the measure at infinity for general trace monoids and not only for irreducible ones. We show that the uniform measure at infinity of a *reducible* trace monoid decomposes as a product of measures on irreducible components—contrasting with uniform distribution at finite horizon. In general, the uniform measure at infinity charges the *infinite* traces of the “largest” components of the monoid, and charges the *finite* traces of the “smallest” components.

Another, different but related, notion of ‘uniform measure’ exists: the Parry measure which is a uniform measure on bi-infinite sequences of an *irreducible* sofic sub-shift [13, 16]. The construction can be applied to trace monoids, defining a ‘uniform measure’ on bi-infinite traces, but only for irreducible trace monoids. Here we focus on single sided infinite traces instead of bi-infinite ones, and this approach allows to relax the irreducibility assumption, and to construct a uniform measure at infinity for a general trace monoid. In case the trace monoid is irreducible, we provide a precise comparison between the Parry measure, restricted to single sided infinite traces, and our uniform measure at infinity. The latter turns out to be a non-stationary version of the former. Another important point is that our approach reveals the combinatorial structure hidden in the uniform measure at infinity (and in the Parry measure).

The outline of the paper is the following. We first focus in a warm-up section (Sect. 2) on the case of two commuting alphabets. Relaxing the commutativity assumption, we arrive to trace monoids in Sect. 3. The purpose of Sect. 4 is twofold: first, to compare the uniform measure with the Parry measure; and second, to examine applications to the uniform sampling of finite traces.

2 Warm-Up: Uniform Measure for Commuting Alphabets

Let A and B be two alphabets and let \mathcal{M} be the product monoid $\mathcal{M} = A^* \times B^*$. The size of $u = (x, y)$ in \mathcal{M} is $|u| = |x| + |y|$. Let $\partial A^* = A^{\mathbb{N}}$ be the set of infinite

A -words, let $\overline{A^*} = A^* \cup A^{\mathbb{N}}$, and similarly for ∂B^* and $\overline{B^*}$. Define:

$$\partial\mathcal{M} = \{(\xi, \zeta) \in \overline{A^*} \times \overline{B^*} : |\xi| + |\zeta| = \infty\}, \quad \overline{\mathcal{M}} = \mathcal{M} \cup \partial\mathcal{M}.$$

Clearly one has $\partial\mathcal{M} = (\overline{A^*} \times \overline{B^*}) - (A^* \times B^*)$ and $\overline{\mathcal{M}} = \overline{A^*} \times \overline{B^*}$. Both $\overline{A^*}$ and $\overline{B^*}$ are equipped with the natural prefix orderings, and $\overline{\mathcal{M}}$ is equipped with the product ordering, denoted by \leq . For $u \in \mathcal{M}$, we put:

$$\uparrow u = \{v \in \overline{\mathcal{M}} : u \leq v\}, \quad \uparrow u = \{\xi \in \partial\mathcal{M} : u \leq \xi\}.$$

Let $p_0 = 1/|A|$ and $q_0 = 1/|B|$. Without loss of generality, we assume that $|A| \geq |B|$, hence $p_0 \leq q_0$.

Lemma 1. *For each real number $p \in (0, p_0]$, there exists a unique probability measure ν_p on $\overline{A^*}$ such that $\nu_p(\uparrow x) = p^{|x|}$ holds for all $x \in A^*$. We have:*

$$\forall p \in (0, p_0) \quad \nu_p(A^*) = 1, \quad \nu_{p_0}(\partial A^*) = 1.$$

The probability measures ν_p in Lemma 1 are called *sub-uniform* measures of parameter p over $\overline{A^*}$. The measure ν_{p_0} is the classical uniform measure on ∂A^* which satisfies $\nu_{p_0}(\uparrow x) = p_0^{|x|}$ for all $x \in A^*$.

For each integer $k \geq 0$, let $\nu_{\mathcal{M}_k}$ denote the uniform distribution on $\mathcal{M}_k = \{(x, y) \in \mathcal{M} : |(x, y)| = k\}$. Since $|A| \geq |B|$, an element $(x, y) \in \mathcal{M}_k$ sampled according to $\nu_{\mathcal{M}_k}$ is more likely to satisfy $|x| \geq |y|$ than the opposite. In the limit, it is natural to expect that infinite elements on the B side are not charged at all, except if $|A| = |B|$. This is made precise in the following result.

Theorem 1. *Let ν_A and ν_B be the sub-uniform measures of parameter $p_0 = 1/|A|$ over $\overline{A^*}$ and $\overline{B^*}$ respectively. The sequence $(\nu_{\mathcal{M}_k})_{k \geq 0}$ converges weakly to the product measure $\nu = \nu_A \otimes \nu_B$.*

We have: $\nu(\uparrow(x, y)) = p_0^{|x|+|y|}$ for all $(x, y) \in \mathcal{M}$; and $\nu(\partial A^ \times B^*) = 1$ if $|A| > |B|$, whereas $\nu(\partial A^* \times \partial B^*) = 1$ if $|A| = |B|$.*

We say that the measure ν described in Theorem 1 is the *uniform* measure on $\partial\mathcal{M}$. We have the following “realization” result for ν .

Theorem 2. *Let $(a_n)_{n \in \mathbb{N}}$ be a sequence of i.i.d. and uniform random variables (r.v.) over A . Let b_0 be a r.v. over $B \cup \{1_{B^*}\}$, where 1_{B^*} is the identity element of B^* , and with the following law:*

$$\forall b \in B \quad \mathbb{P}(b_0 = b) = p_0 = 1/|A|, \quad \mathbb{P}(b_0 = 1_{B^*}) = 1 - p_0/q_0 = 1 - |B|/|A|.$$

Consider $(b_n)_{n \in \mathbb{N}}$ sampled independently in $B \cup \{1_{B^}\}$, each b_n with the same law as b_0 , but only until it reaches 1_{B^*} , after which b_n is constant equal to 1_{B^*} . Finally, set $u_k \in \mathcal{M}$ for all integers $k \geq 0$ by:*

$$x_k = a_0 \cdot \dots \cdot a_{k-1} \in A^*, \quad y_k = b_0 \cdot \dots \cdot b_{k-1} \in B^*, \quad u_k = (x_k, y_k) \in \mathcal{M}.$$

Then $(u_k)_{k \in \mathbb{N}}$ converges in law towards ν . Furthermore, the random variable $\bigvee_{k \geq 0} u_k \in \partial\mathcal{M}$ is distributed according to ν .

Observe that 1_{B^*} will eventually appear in the sequence $(b_n)_{n \in \mathbb{N}}$ with probability 1 if and only if $p_0 < q_0$. In this case, $(y_n)_{n \in \mathbb{N}}$ is eventually equal to a constant element of B^* with probability 1. This is consistent with Theorem 1. Observe also that $(a_n, b_n)_{n \in \mathbb{N}}$ forms a product Markov chain on $A \times (B \cup \{1_{B^*}\})$.

Both results stated in Theorems 1 and 2 are particular cases of corresponding results for trace monoids, as we will see next.

3 Uniform and Sub-uniform Measures for Trace Monoids

Basics on Trace Monoids. Let A be a finite alphabet equipped with an irreflexive and symmetric relation $I \subseteq A \times A$, called an *independence relation*. The pair (A, I) is called an *independence pair*. Let \mathcal{I} be the congruence relation on the free monoid A^* generated by the collection of pairs (ab, ba) for (a, b) ranging over I . The *trace monoid* $\mathcal{M} = \mathcal{M}(A, I)$ is defined as the quotient monoid $\mathcal{M} = A^*/\mathcal{I}$, see [6, 8, 18]. The elements of \mathcal{M} are called *traces*. The identity element in the monoid is called the *empty trace*, denoted “ $1_{\mathcal{M}}$ ”, and the concatenation is denoted with the dot “ \cdot ”.

The length of a trace u is well defined as the length of any of its representative words and is denoted by $|u|$. The left divisibility relation on \mathcal{M} is a partial order, denoted by “ \leq ” and defined by: $u \leq v \iff \exists w \ v = u \cdot w$.

An intuitive representation of traces is given by Viennot’s *heap of pieces* interpretation of a trace monoid [18]. We illustrate in Fig. 1 the heap of pieces interpretation for the monoid $\mathcal{M}(A, I)$ with $A = \{a, b, c\}$ and $I = \{(a, b), (b, a)\}$.

The length of traces corresponds to the number of pieces in a heap. The relation $u \leq v$ corresponds to u being seen at bottom as a sub-heap of heap v .

The product monoid $A^* \times B^*$ from Sect. 2 is isomorphic to the trace monoid $\mathcal{M}(\Sigma, I)$, where $\Sigma = A \cup B$ with A and B being considered as disjoint, and $I = (A \times B) \cup (B \times A)$.

Cliques and Height of Traces. Recall that a *clique* of a graph is a complete subgraph (by convention, the empty graph is a clique). We may view (A, I) as a graph. Given a clique c of (A, I) , the product $a_1 \dots a_j \in \mathcal{M}$ is independent of the enumeration (a_1, \dots, a_j) of the vertices composing c . We say that $a_1 \dots a_j$ is a *clique* of \mathcal{M} . Let \mathcal{C} denote the set of cliques, including the empty clique $1_{\mathcal{M}}$. As heaps of pieces, cliques correspond to flat heaps, or horizontal layers.

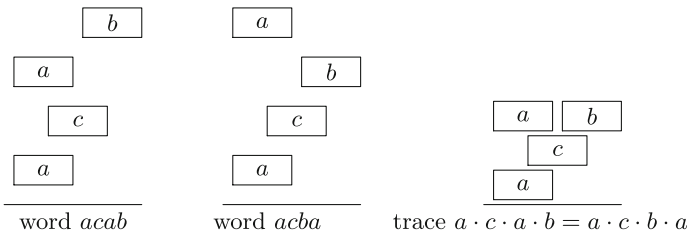


Fig. 1. Two congruent words and the resulting heap (trace)

Traces are known to admit a canonical normal form, defined as follows [6]. Say that two non-empty cliques c, c' are *Cartier-Foata admissible*, denoted by $c \rightarrow c'$, whenever they satisfy: $\forall a \in c' \exists b \in c (b, a) \notin I$. For every non empty trace $u \in \mathcal{M}$, there exists a unique integer $n > 0$ and a unique sequence (c_1, \dots, c_n) of non-empty cliques such that: (1) $u = c_1 \dots c_n$; and (2) $c_i \rightarrow c_{i+1}$ holds for all $i \in \{1, \dots, n-1\}$. The integer n is called the *height* of u , denoted by $n = \tau(u)$. By convention, we put $\tau(1_{\mathcal{M}}) = 0$. The sequence (c_1, \dots, c_n) is called the *Cartier-Foata normal form* or *decomposition* of u . In the heap interpretation, the normal form corresponds to the sequence of horizontal layers that compose a heap u , and the height $\tau(u)$ corresponds to the number of horizontal layers.

A useful device is the notion of *topping* of traces, defined as follows: for each integer $n \geq 0$, the n -*topping* is the mapping $\kappa_n : \mathcal{M} \rightarrow \mathcal{M}$ defined by $\kappa_n(u) = c_1 \dots c_n$, where $c_1 \rightarrow \dots \rightarrow c_p$ is the Cartier-Foata decomposition of u , and where $c_i = 1_{\mathcal{M}}$ if $i > p$.

Boundary. Elementary Cylinders. Let $\mathfrak{C} = \mathcal{C} \setminus \{1_{\mathcal{M}}\}$ denote the set of non-empty cliques. Traces of \mathcal{M} are in bijection with finite paths of the automaton $(\mathfrak{C}, \rightarrow)$, where all states are both initial and final. Denote by $\partial\mathcal{M}$ the set of infinite paths in the automaton $(\mathfrak{C}, \rightarrow)$. We call $\partial\mathcal{M}$ the *boundary at infinity*, or simply the *boundary*, of monoid \mathcal{M} , and we put $\overline{\mathcal{M}} = \mathcal{M} \cup \partial\mathcal{M}$. Elements of $\partial\mathcal{M}$ are called *infinite traces*, and, by contrast, elements of \mathcal{M} might be called *finite traces*.

By construction, an infinite trace is given as an infinite sequence $\xi = (c_1, c_2, \dots)$ of non-empty cliques such that $c_i \rightarrow c_{i+1}$ holds for all integers $i \geq 1$. Note that the topping operations extend naturally to $\kappa_n : \overline{\mathcal{M}} \rightarrow \mathcal{M}$, defined by $\kappa_n(\xi) = c_1 \dots c_n$, for $\xi = (c_1, c_2, \dots)$.

We wish to extend the partial order relation \leq from \mathcal{M} to $\overline{\mathcal{M}}$. For this, we first recall the following result [2, Cor. 4.2]: for $u, v \in \mathcal{M}$, if $n = \tau(u)$, then $u \leq v \iff u \leq \kappa_n(v)$. Henceforth, we put $\zeta \leq \xi \iff \forall n \geq 0 \kappa_n(\zeta) \leq \kappa_n(\xi)$ for $\zeta, \xi \in \overline{\mathcal{M}}$, consistently with the previous definition in case $\zeta, \xi \in \mathcal{M}$. This order is coarser than the prefix ordering on sequences of cliques.

For each $u \in \mathcal{M}$, we define two kinds of *elementary cylinders of base u*:

$$\uparrow u = \{\xi \in \partial\mathcal{M} : u \leq \xi\} \subseteq \partial\mathcal{M}, \quad \uparrow u = \{v \in \overline{\mathcal{M}} : u \leq v\} \subseteq \overline{\mathcal{M}}. \quad (1)$$

The set \mathcal{M} being countable, it is equipped with the discrete topology. The set $\overline{\mathcal{M}}$ is a compactification of \mathcal{M} , when equipped with the topology generated by the opens of \mathcal{M} and all cylinders $\uparrow u$, for u ranging over \mathcal{M} . This makes $\overline{\mathcal{M}}$ a metrisable compact space [1]. The set $\partial\mathcal{M}$ is a closed subset of $\overline{\mathcal{M}}$. The induced topology on $\partial\mathcal{M}$ is generated by the family of cylinders $\uparrow u$, for u ranging over \mathcal{M} . Finally, both spaces are equipped with their respective Borel σ -algebras, \mathfrak{F} on \mathcal{M} and $\overline{\mathfrak{F}}$ on $\overline{\mathcal{M}}$; the σ -algebra on each space is generated by the corresponding family of cylinders.

Möbius polynomial. Principal Root. Sub-uniform Measures. We recall [6, 18] the definitions of the *Möbius polynomial* $\mu_{\mathcal{M}}(X)$ and of the *growth series* $G(X)$ associated to \mathcal{M} :

$$\mu_{\mathcal{M}}(X) = \sum_{c \in \mathcal{C}} (-1)^{|c|} X^{|c|}, \quad G(X) = \sum_{u \in \mathcal{M}} X^{|u|} = \sum_{n \geq 0} \lambda_{\mathcal{M}}(n) X^n, \quad (2)$$

where $\lambda_{\mathcal{M}}(n) = \#\{x \in \mathcal{M} : |x| = n\}$. It is known that $G(X)$ is rational, inverse of the Möbius polynomial:

$$G(X) = 1/\mu_{\mathcal{M}}(X).$$

It is also known [7, 14] that $\mu_{\mathcal{M}}(X)$ has a unique root of smallest modulus, say p_0 , which lies in the real interval $(0, 1)$ if $|A| > 1$ (the case $|A| = 1$ is trivial). The root p_0 will be called the *principal root* of $\mu_{\mathcal{M}}$, or simply of \mathcal{M} .

The following result, to be compared with Lemma 1, adapts the so-called Patterson-Sullivan construction from geometric group theory. The compactness of $\overline{\mathcal{M}}$ is an essential ingredient of the proof for the case $p = p_0$, based on classical results from Functional Analysis.

Theorem 3. *For each $p \in (0, p_0]$, where p_0 is the principal root of \mathcal{M} , there exists a unique probability measure ν_p on $(\overline{\mathcal{M}}, \overline{\mathfrak{F}})$ such that $\nu_p(\uparrow x) = p^{|x|}$ holds for all $x \in \mathcal{M}$. On the one hand, if $p < p_0$, then ν_p is concentrated on \mathcal{M} , and is given by:*

$$\forall x \in \mathcal{M} \quad \nu_p(\{x\}) = p^{|x|}/G(p). \tag{3}$$

On the other hand, ν_{p_0} is concentrated on the boundary, hence $\nu_{p_0}(\partial\mathcal{M}) = 1$. In this case, $\nu_{p_0}(\uparrow x) = p_0^{|x|}$ holds for all $x \in \mathcal{M}$.

Definition 1. *The measures ν_p on $\overline{\mathcal{M}}$ described in Theorem 3 are called sub-uniform measures of parameter p . The measure ν_{p_0} is called the uniform measure on $\partial\mathcal{M}$.*

The following result relates the uniform measure on the boundary with the sequence $\nu_{\mathcal{M}_k}$ of uniform distributions over the sets $\mathcal{M}_k = \{x \in \mathcal{M} : |x| = k\}$.

Theorem 4. *Let \mathcal{M} be a trace monoid, of principal root p_0 . The sequence of uniform distributions $(\nu_{\mathcal{M}_k})_{k \geq 0}$ converges weakly toward the uniform measure ν_{p_0} on $\partial\mathcal{M}$.*

Anticipating on Theorem 5 below, Theorem 4 above has the following concrete consequence. Fix an integer $j \geq 1$, and draw traces of length k uniformly at random, with k arbitrarily large. Then the j first cliques of the trace obtained approximately behave as if they were a Markov chain (C_1, \dots, C_j) ; and the larger k , the better the approximation. Conversely, how this can be exploited for random generation purposes, is the topic of Sect. 4.

Irreducibility and Irreducible Components. Generators of a trace monoid only have partial commutativity properties. The following definition isolates the parts of the alphabet that enjoy full commutativity.

Definition 2. *Let (A, I) be an independence pair. The associated dependence pair is (A, D) where $D = (A \times A) \setminus I$. The connected components of the graph (A, D) are called the irreducible components of $\mathcal{M} = \mathcal{M}(A, I)$. To each of these irreducible component A' is associated the independence relation $I' = I \cap (A' \times A')$. The corresponding trace monoids $\mathcal{M}' = \mathcal{M}(A', I')$ are called the irreducible components of the trace monoid \mathcal{M} . If (A, D) is connected, then \mathcal{M} is said to be irreducible.*

Direct products of trace monoids are trace monoids themselves. More precisely, the following result holds.

Proposition 1. *Let $\mathcal{M} = \mathcal{M}(A, I)$ be a trace monoid. Then \mathcal{M} is the direct product of its irreducible components. As a measurable space and as a topological space, $\overline{\mathcal{M}}$ is the product of the $\overline{\mathcal{M}'}$, where \mathcal{M}' ranges over the irreducible components of \mathcal{M} . The Möbius polynomial $\mu_{\mathcal{M}}(X)$ is the product of the Möbius polynomials $\mu_{\mathcal{M}'}(X)$, for \mathcal{M}' ranging over the irreducible components of \mathcal{M} .*

The sets $\mathcal{M}_k = \{x \in \mathcal{M} : |x| = k\}$ do not enjoy a product decomposition with respect to irreducible components of \mathcal{M} , hence neither do the uniform distributions $\nu_{\mathcal{M}_k}$ over \mathcal{M}_k . By contrast, sub-uniform measures have a product decomposition, as stated below.

Proposition 2. *Let \mathcal{M} be a trace monoid, of principal root p_0 , and let ν_p be a sub-uniform measure on $\overline{\mathcal{M}}$ with $p \leq p_0$. Then ν_p is the product of measures ν' on each of the $\overline{\mathcal{M}'}$, for \mathcal{M}' ranging over the irreducible components of \mathcal{M} . The measures ν' are all sub-uniform measures on $\overline{\mathcal{M}'}$ of the same parameter p .*

It follows from Proposition 1 that the principal root of a trace monoid \mathcal{M} is the smallest among the principal roots of its irreducible components. As a consequence of Proposition 2, the uniform measure is a product of *sub-uniform* measures ν' over the irreducible components \mathcal{M}' of \mathcal{M} . By Theorem 3, each ν' is either concentrated on \mathcal{M}' if the principal root p' of \mathcal{M}' satisfies $p' > p_0$, or concentrated on $\partial\mathcal{M}'$ if $p' = p_0$. Note that at least one of these sub-uniform measures is actually *uniform* on the irreducible component.

Realization of Uniform and Sub-uniform Measures. The characterization of the uniform measure by $\nu(\uparrow x) = p_0^{|x|}$ (see Theorem 3) does not provide an obvious recursive procedure for an algorithmic approximation of ν -generated samples on $\partial\mathcal{M}$. Since the uniform measure ν is, according to Proposition 2, a product of sub-uniform measures, it is enough to focus on the algorithmic sampling of *sub-uniform measures on irreducible trace monoids*.

Hence, let \mathcal{M} be an irreducible trace monoid, of principal root p_0 , and let $\overline{\mathcal{M}}$ be equipped with a sub-uniform measure ν_p with $p \leq p_0$. Recall from Theorem 3 that ν_p is either concentrated on \mathcal{M} or on $\partial\mathcal{M}$ according to whether $p < p_0$ or $p = p_0$.

Elements of \mathcal{M} are given as finite paths in the graph $(\mathcal{C}, \rightarrow)$, whereas elements of $\partial\mathcal{M}$ are given as infinite paths in $(\mathcal{C}, \rightarrow)$. In order to have a unified presentation of both spaces, we use the following technical trick: instead of considering the graph of *non empty* cliques $(\mathcal{C}, \rightarrow)$, we use the graph of *all* cliques $(\mathcal{C}, \rightarrow)$, including the empty clique. We keep the same definition of the Cartier-Foata relation ‘ \rightarrow ’ (see above). Note that $c \rightarrow 1_{\mathcal{M}}$ then holds for every clique $c \in \mathcal{C}$, whereas $1_{\mathcal{M}} \rightarrow c$ holds if and only if $c = 1_{\mathcal{M}}$. Hence $1_{\mathcal{M}}$ is an absorbing state in $(\mathcal{C}, \rightarrow)$. Any path in $(\mathcal{C}, \rightarrow)$, either finite or infinite, now corresponds to a unique infinite path in $(\mathcal{C}, \rightarrow)$. If the original path $(c_k)_{1 \leq k \leq N}$ is finite, the corresponding infinite path $(c'_k)_{k \geq 1}$ in $(\mathcal{C}, \rightarrow)$ is defined by $c'_k = c_k$ for $1 \leq k \leq N$ and $c'_k = 1_{\mathcal{M}}$ for all $k > N$.

For each trace $\xi \in \overline{\mathcal{M}}$, either finite or infinite, let $(C_k)_{k \geq 1}$ be the infinite sequence of cliques corresponding to the infinite path in $(\mathcal{C}, \rightarrow)$ associated with ξ . The sequence $(C_k)_{k \geq 1}$ is a random sequence of cliques; its characterization under a sub-uniform measure ν_p is the topic of next result.

Theorem 5. *Let \mathcal{M} be an irreducible trace monoid of principal root p_0 . Then, with respect to the sub-uniform measure ν_p on $\overline{\mathcal{M}}$, with $0 < p \leq p_0$, the sequence of random cliques $(C_k)_{k \geq 1}$ is a Markov chain with state space \mathcal{C} .*

Let $g, h : \mathcal{C} \rightarrow \mathbb{R}$ be the functions defined by:

$$h(c) = \sum_{c' \in \mathcal{C} : c' \geq c} (-1)^{|c'| - |c|} p^{|c'|}, \quad g(c) = h(c)/p^{|c|}. \quad (4)$$

Then $(h(c))_{c \in \mathcal{C}}$ is a probability vector over \mathcal{C} , which is the distribution of the initial clique C_1 . This vector is positive on \mathcal{C} , and $h(1_{\mathcal{M}}) > 0$ if and only if $p < p_0$. The transition matrix of the chain, say $P = (P_{c,c'})_{(c,c') \in \mathcal{C} \times \mathcal{C}}$, is:

$$P_{c,c'} = \begin{cases} 0, & \text{if } c \rightarrow c' \text{ does not hold,} \\ h(c')/g(c), & \text{if } c \rightarrow c' \text{ holds,} \end{cases} \quad (5)$$

with the line $(P_{1_{\mathcal{M}},c'})_{c' \in \mathcal{C}}$ corresponding to the empty clique undefined if $p = p_0$.

Conversely, if $p \leq p_0$, and if $(C_k)_{k \geq 1}$ is a Markov chain on \mathcal{C} if $p < p_0$, respectively on \mathcal{C} if $p = p_0$, with initial distribution h defined in (4) and with transition matrix P defined in (5), and if $Y_k = C_1 \cdot \dots \cdot C_k$, then $(Y_k)_{k \geq 1}$ converges weakly towards the sub-uniform measure ν_p . Furthermore, the law of the random trace $C_1 \cdot C_2 \cdot \dots = \bigvee_{k \geq 1} Y_k \in \overline{\mathcal{M}}$ is the probability measure ν_p on $\overline{\mathcal{M}}$.

Theorem 5 for $p = p_0$ already appears in [2]. Note: the function $h : \mathcal{C} \rightarrow \mathbb{R}$ defined in (4) is the Möbius transform in the sense of Rota [3, 17] of the function $f : c \in \mathcal{C} \mapsto p^{|c|}$; see [2] for more emphasis on this point of view.

As expected, we recover the results of Sect. 2 in the case of two commuting alphabets A and B with $|A| > |B|$. Indeed, by Proposition 2 and Theorem 5, the cliques $(C_k)_{k \geq 1}$ form a product of two Markov chains: one on A (non empty cliques of A^*) and the other one on $B \cup \{1_{B^*}\}$ (cliques of B^* , including the empty one).

4 Uniform Generation of Finite Traces

We have introduced in Definition 1 a notion of uniform measure on the boundary of a trace monoid. This measure is characterized by its values on cylinders in Theorem 3, as the weak limit of uniform distributions in Proposition 2, and through the associated Cartier-Foata probabilistic process in Theorem 5.

Because of the existence of the Cartier-Foata normal form of traces, the combinatorics of a trace monoid is entirely contained in the Cartier-Foata automaton, either $(\mathcal{C}, \rightarrow)$ or $(\mathcal{C}, \rightarrow)$. Looking at the Cartier-Foata automaton, say $(\mathcal{C}, \rightarrow)$ on non empty-cliques, as generating a sub-shift of finite type, it is interesting to investigate the associated notion of uniform measure ‘à la Parry’ [13, 15, 16],

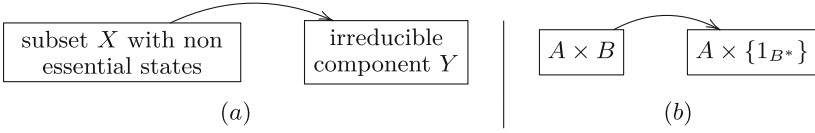


Fig. 2. (a) Illustration of a reducible system (b) Cartier-Foata automaton on non-empty cliques of $A^* \times B^*$ generating the uniform measure on $\partial(A^* \times B^*)$ if $|A| > |B|$

and to compare it with the uniform measure on the boundary previously introduced. This comparison between the two notions of uniform measures will enlighten the forthcoming discussion on uniform generation of finite traces.

Uniform Measure on the Boundary Versus Parry Measure. The Parry measure associated with an *irreducible* sub-shift of finite type is formally defined as the unique measure of maximal entropy on *bi-infinite* admissible sequences of states of the sub-shift. It corresponds intuitively to the “uniform measure” on such bi-infinite paths (see, e.g., [15]).

The Parry measure is only defined for *irreducible* sub-shifts for good reasons. Indeed, if a sub-shift has, say, two parts X and Y , with Y an irreducible component and such that going from X to Y is possible but not the other way around as in Fig. 2(a), then one cannot define a “uniform measure” on bi-infinite paths (it should put mass on paths spending an infinite amount of time both in X and Y and be stationary, which is impossible). On the other hand, considering a uniform measure on one-sided infinite sequences on such a compound system makes perfect sense. This is the case, for instance, of the Cartier-Foata sub-shift associated to the reducible trace monoids $A^* \times B^*$ with $|A| > |B|$ studied in Sect. 2: see Fig. 2(b).

For a general trace monoid \mathcal{M} , the associated sub-shift $(\mathfrak{C}, \rightarrow)$ is irreducible if and only if the monoid \mathcal{M} is irreducible in the sense of Definition 2 (a well-known result: see for instance [14, Lemma 3.2]). Therefore the comparison between the uniform measure on the boundary, and the Parry measure, only makes sense in this case.

Hence, let \mathcal{M} be an irreducible trace monoid, of principal root p_0 . In order to take into account the length of cliques in the construction of the Parry measure, we consider the weighted incidence matrix $B = (B_{x,y})_{(x,y) \in \mathfrak{C} \times \mathfrak{C}}$ defined by $B_{x,y} = p_0^{|y|}$ if $x \rightarrow y$ holds and by $B_{x,y} = 0$ if $x \rightarrow y$ does not hold.

Lemma 2. *The non-negative matrix B has spectral radius 1. The vector $g = (g(c))_{c \in \mathfrak{C}}$ defined by $g(c) = \sum_{c' \in \mathfrak{C} : c \rightarrow c'} h(c')$ for $c \in \mathfrak{C}$, where h has been defined in (4), is B -invariant on the right: $Bg = g$.*

Define the matrix $C = (C_{c,c'})_{(c,c') \in \mathfrak{C} \times \mathfrak{C}}$ by:

$$\forall c, c' \in \mathfrak{C} \quad C_{c,c'} = B_{c,c'} g(c') / g(c). \quad (6)$$

Since g is right invariant for B , it follows that C is stochastic. Classically, the Parry measure on bi-infinite paths in $(\mathfrak{C}, \rightarrow)$ is the stationary Markovian measure of transition matrix C .

Proposition 3. *The matrix C defined in (6) coincides with the transition matrix P defined in Theorem 5 for $p = p_0$, and restricted to $\mathfrak{C} \times \mathfrak{C}$.*

Proposition 3 asserts that the Markov chain associated with the Parry measure has the same transition matrix as the probabilistic process on non-empty cliques generated by the uniform measure on the boundary. But the Parry measure is stationary whereas the uniform measure ν is not. Indeed, the initial distribution of the Markov measure ν is $h : \mathfrak{C} \rightarrow \mathbb{R}$, which does *not* coincide with the stationary measure of the chain (except in the trivial case of a free monoid).

To summarize: the notion of uniform measure on the boundary is adapted to one-sided infinite heaps, independently of the irreducibility of the trace monoid under consideration. If the monoid is irreducible, there is a notion of uniform measure on two-sided infinite heaps, which correspond to a weighted Parry measure. Considering the projection of this Parry measure to one-sided infinite heaps, and conditionally on a given initial clique, it coincides with the uniform measure at infinity since they share the same transition matrix. But the two measures globally differ since their initial measures differ.

Uniform Generation of Finite Traces, 0. The Parry measure is a standard tool for a special type of uniform generation. Indeed, it provides an algorithmic way of sampling finite sequences of a fixed length k , and uniformly *if the first and the last letters of the sequence are given*. In our framework, besides the fact that the Parry measure is only defined for an irreducible trace monoid, it also misses the primary target of generating finite traces of a given length k among *all traces* of length k .

Uniform Generation of Finite Traces, 1. Consider the problem, given a fixed integer $k > 1$ and a trace monoid $\mathcal{M} = \mathcal{M}(A, I)$, of designing a randomized algorithm which produces a trace $x \in \mathcal{M}$ of length k , uniformly among traces of length k . Sub-uniform measures on the trace monoid \mathcal{M} allow to adapt to our framework the technique of Boltzmann samplers [10] for solving this problem.

Consider a parameter $p \in (0, p_0)$, where p_0 is the principal root of \mathcal{M} , and let $\xi \in \mathcal{M}$ be sampled according to the sub-uniform measure ν_p . We have indeed $|\xi| < \infty$ with probability 1 by Theorem 3. Furthermore, Proposition 2 shows that ν_p decomposes as a product of sub-uniform measures of the same parameter p , over the irreducible components of \mathcal{M} . For each component, sampling is done through usual Markov chain generation techniques since both the initial measure and the transition matrix of the chain of cliques are explicitly known by Theorem 5.

The algorithm is then the following: if $|\xi| = k$, then keep ξ ; otherwise, reject ξ and sample another trace. This eventually produces a random trace of length k , uniformly distributed in \mathcal{M}_k ; since ν_p is a weighted sum of all $\nu_{\mathcal{M}_k}$, as shown by the expression (3).

As usual, the optimal parameter p , for which the rejection probability is the lowest, is such that: $\mathbb{E}_{\nu_p}|\xi| = k$, where $\mathbb{E}_{\nu_p}(\cdot)$ denotes the expectation with respect to ν_p . Ordinary computations show that $\mathbb{E}_{\nu_p}|\xi|$ is related to the derivative of the growth function by $\mathbb{E}_{\nu_p}|\xi| = pG'(p)/G(p) = -p\mu'_{\mathcal{M}}(p)/\mu_{\mathcal{M}}(p)$; providing an explicit equation

$$k\mu_{\mathcal{M}}(p) + p\mu'_{\mathcal{M}}(p) = 0,$$

to be numerically solved in p .

Unfortunately, the rejection probability approaches 1 exponentially fast as k increases, making the algorithm less and less efficient. A standard way to overcome this difficulty would be to consider *approximate sampling* [10], consisting in sampling traces of length approximately k .

Uniform Generation of Finite Traces, 2: Evaluating an Average Cost.

Uniform generation is often done in order to evaluate the expected value of a cost function. For this purpose, a more direct approach in our framework is based on an exact integration formula given in Theorem 6 below.

Let $\phi : \mathcal{M}_k \rightarrow \mathbb{R}$ be a cost function, and consider the problem of evaluating the expectation $\mathbb{E}_{\nu_{\mathcal{M}_k}}(\phi)$, for a fixed integer k . For each integer $k \geq 0$, let:

$$\mathcal{M}_k = \{x \in \mathcal{M} : |x| = k\}, \quad \lambda_{\mathcal{M}}(k) = \#\mathcal{M}_k, \quad \mathcal{M}_{(k)} = \{x \in \mathcal{M} : \tau(x) = k\}.$$

To each function $\phi : \mathcal{M}_k \rightarrow \mathbb{R}$ defined on traces of *length* k , we associate a function $\bar{\phi} : \mathcal{M}_{(k)} \rightarrow \mathbb{R}$ defined on traces of *height* k , as follows:

$$\forall x \in \mathcal{M}_{(k)} \quad \bar{\phi}(x) = \sum_{y \in \mathcal{M}_k : y \leq x} \phi(y). \quad (7)$$

Theorem 6. *Let $\bar{\phi} : \mathcal{M}_{(k)} \rightarrow \mathbb{R}$ be defined as in (7). Then the following equality holds between the expectation with respect to the uniform distribution $\nu_{\mathcal{M}_k}$ on \mathcal{M}_k on the one hand, and the expectation with respect to the uniform measure ν on $\partial\mathcal{M}$ on the other hand (whether \mathcal{M} is irreducible or not):*

$$\mathbb{E}_{\nu_{\mathcal{M}_k}} \phi = (p_0^k \cdot \lambda_{\mathcal{M}}(k))^{-1} \cdot \mathbb{E}_{\nu} \bar{\phi}(C_1 \cdot \dots \cdot C_k). \quad (8)$$

The generation of $(C_k)_{k \geq 1}$ enables us to evaluate $\mathbb{E}_{\nu} \bar{\phi}(C_1 \cdot \dots \cdot C_k)$ for any integer k , provided the function $\bar{\phi}$ can be efficiently computed. In turn, this directly depends on the numbers $\theta_k(x) = \#\{y \in \mathcal{M}_k : y \leq x\}$ of terms in the sum (7) defining $\bar{\phi}(x)$. The numbers $\theta_k(x)$ might be arbitrary large; for instance $\theta_k((a \cdot b)^k) = k + 1$ for $(a, b) \in I$. However we have the following result.

Lemma 3. *Assume that \mathcal{M} is irreducible. Then, there exists $C > 0$ such that:*

$$\mathbb{E}_{\nu} \theta_k(C_1 \cdot \dots \cdot C_k) \leq C.$$

To see this, apply (8) to the constant function $\phi = 1$ on \mathcal{M}_k , whose associated function is $\bar{\phi} = \theta_k$ on $\mathcal{M}_{(k)}$, to obtain:

$$\mathbb{E}_{\nu} \theta_k(C_1 \cdot \dots \cdot C_k) = p_0^k \cdot \lambda_{\mathcal{M}}(k). \quad (9)$$

The terms $\lambda_{\mathcal{M}}(k)$, coefficients of the growth series $G(X) = 1/\mu_{\mathcal{M}}(X)$, are asymptotically equivalent to Cp_0^{-k} for some constants $C > 0$ if \mathcal{M} is irreducible [14]. The result in Lemma 3 follows.

Applying usual techniques [4] to specifically retrieve all traces $y \leq x$ of length $k = \tau(x)$ is feasible in time $O(k)$ in average and allows to compute $\bar{\phi}(x)$, and consequently to estimate the expectation $\mathbb{E}_\nu \bar{\phi}(C_1 \dots C_k)$ via Markov chain sampling and a Monte-Carlo algorithm.

By (9), applying the same estimation technique to the function $\phi = 1$ yields an estimate for the normalization factor $p_0^k \cdot \lambda_{\mathcal{M}}(k)$. In passing, this also yields a Monte-Carlo estimate for the number $\lambda_{\mathcal{M}}(k)$. All together, we are thus able to estimate with an arbitrary precision both terms in the right hand member of (8), hence yielding an accurate estimation of $\mathbb{E}_{\nu_{\mathcal{M}_k}} \phi$.

To summarize: generating the first k layers of traces under the uniform measure on the boundary allows to compute the *expectation* of an arbitrary computable cost function $\phi : \mathcal{M}_k \rightarrow \mathbb{R}$, if \mathcal{M} is irreducible. The same applies at the cost of a greater complexity if \mathcal{M} is not irreducible.

References

1. Abbes, S., Keimel, K.: Projective topology on bifinite domains and applications. *Theoret. Comput. Sci.* **365**(3), 171–183 (2006)
2. Abbes, S., Mairesse, J.: Uniform and Bernoulli measures on the boundary of trace monoids. *J. Combin. Theory Ser. A* **135**, 201–236 (2015)
3. Aigner, M.: *A Course in Enumeration*. Springer, Heidelberg (2007)
4. Bertoni, A., Goldwurm, M., Mauri, G., Sabadini, N.: Counting techniques for inclusion, equivalence and membership problems. In: *The Book of Traces*, pp. 131–163. World Scientific (1994)
5. Bodini, O., Genitrini, A., Peschanski, F.: Enumeration and random generation of concurrent computations. In: *Proceedings of AofA 2012*, pp. 83–96. DMTCS (2012)
6. Cartier, P., Foata, D.: *Problèmes combinatoires de commutation et réarrangements*. Lecture Notes in Math. Springer, Heidelberg (1969)
7. Csikvári, P.: Note on the smallest root of the independence polynomial. *Combin. Probab. Comput.* **22**(1), 1–8 (2013)
8. Diekert, V. (ed.): *Combinatorics on Traces*. LNCS, vol. 454. Springer, Heidelberg (1990)
9. Diekert, V., Rozenberg, G. (eds.): *The Book of Traces*. World Scientific, Singapore (1995)
10. Duchon, P., et al.: Boltzmann samplers for the random generation of combinatorial structures. *Combin. Probab. Comput.* **13**, 577–625 (2004)
11. Flajolet, P., Zimmermann, P., Van Cutsem, B.: Calculus for the random generation of labelled combinatorial structures. *Theoret. Comp. Sci.* **218**(2), 233–248 (1994)
12. Jerrum, M.: *Counting, Sampling and Integrating: Algorithms and Complexity*. Springer, Berlin (2013)
13. Kitchens, B.P.: *Symbolic Dynamics. One-sided, Two-sided and Countable State Markov Shifts*. Springer, Berlin (1998)
14. Krob, D., Mairesse, J., Michos, I.: Computing the average parallelism in trace monoids. *Discrete Math.* **273**, 131–162 (2003)
15. Lind, D., Marcus, B.: *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, Cambridge (1995)
16. Parry, W.: Intrinsic Markov chains. *Trans. Amer. Math. Soc.* **112**(1), 55–66 (1964)

17. Rota, G.-C.: On the foundations of combinatorial theory I. Theory of Möbius functions. *Z. Wahrscheinlichkeitstheorie* **2**, 340–368 (1964)
18. Viennot, G.X.: Heaps of pieces, i : basic definitions and combinatorial lemmas. In: Labelle, G., Leroux, P. (eds.) *Combinatoire Énumérative*. LNCS, vol. 1234, pp. 321–350. Springer, Heidelberg (2006)

When Are Prime Formulae Characteristic?

L. Aceto^(✉), D. Della Monica, I. Fábregas, and A. Ingólfssdóttir

ICE-TCS, School of Computer Science, Reykjavik University, Reykjavik, Iceland
{luca,dariodm,fabregas,annai}@ru.is

Abstract. In the setting of the modal logic that characterizes modal refinement over modal transition systems, Boudol and Larsen showed that the formulae for which model checking can be reduced to preorder checking, that is, the characteristic formulae, are exactly the consistent and prime ones. This paper presents general, sufficient conditions guaranteeing that characteristic formulae are exactly the consistent and prime ones. It is shown that the given conditions apply to the logics characterizing all the semantics in van Glabbeek's branching-time spectrum.

1 Introduction

Model checking and equivalence/preorder checking are the two main approaches to the computer-aided verification of reactive systems [3,6]. In model checking, one typically describes the behaviour of a computing system using a state-transition model, such as a labelled transition system [11], and specifications of properties systems should exhibit are expressed using some modal or temporal logic. In this approach, system verification amounts to checking whether a system is a model of the formulae describing a given specification. When using equivalence/preorder checking instead, systems and their specifications are both expressed in the same state-machine-based formalism. In this approach, checking whether a system correctly implements its specification amounts to verifying whether the state machines describing them are related by some suitable notion of behavioural equivalence/preorder. (See [8,9] for taxonomic studies of the plethora of behavioural relations that have been considered in the field of concurrency theory.)

A bridge between model checking and equivalence/preorder checking is provided by the notion of *characteristic formula* [10,13]. Intuitively, a characteristic formula provides a complete logical characterization of the behaviour of a process modulo some notion of behavioural equivalence or preorder. At least for finite labelled transition systems, such formulae can be used to reduce equivalence/preorder checking to model checking effectively, and, as argued in [7], this

Research supported by the project 001-ABEL-CM-2013 within the NILS Science and Sustainability Programme, the Spanish project STRONGSOFT TIN2012-39391-C04-04, and the projects *Nominal SOS* (project nr. 141558-051) and *Decidability and Expressiveness for Interval Temporal Logics* (project nr. 130802-051) of the Icelandic Research Fund.

approach has better complexity than known algorithms for preorder checking. A natural question to ask is for what kinds of logical specifications model checking can be reduced to establishing a behavioural relation between an implementation and a labelled transition system that suitably encodes the specification. To the best of our knowledge, this question was first addressed by Boudol and Larsen, who showed in [5] that, in the context of the modal logic that characterizes modal refinement over modal transition systems, the formulae that are “graphically representable” (that is, the ones that are characteristic for some process) are exactly the consistent and prime ones. (A formula is *prime* if whenever it implies a disjunction of two formulae, it implies one of the disjuncts.) A similar result is given in [2] in the setting of covariant-contravariant simulation. Moreover, each formula in the logics considered in [2, 5] can be “graphically represented” by a (possibly empty) finite set of processes.

To our mind, those are very pleasing results that show the very close connection between logical and behavioural approaches to verification in two specific settings. But, how general are they? Do similar results hold for the plethora of other process semantics and their modal characterizations studied in the literature? And, if so, are there general sufficient conditions guaranteeing that characteristic formulae are exactly the consistent and prime ones? The aim of this article is to provide answers to those questions.

From a methodological perspective, we follow a purely logical approach towards the characterization of process semantics, which allows us to work in an abstract and very general setting (described in Sect. 2): instead of investigating each behavioural semantics separately, we define a process semantics as the preorder induced by some logic, i.e. a process p is smaller than a process q if the set of logical properties of p is strictly included in that of q . By investigating preorders defined in this way, we can identify common properties for all logically characterized preorders, and thus we are able to give a general recipe to logically characterize processes by means of consistent and prime formulae (*characterization by primality*). The first piece of our characterization by primality result consists in showing that characteristic formulae are always consistent and prime (Theorem 1). This result was already proven for specific semantics [2, 5], and we generalise it here to every logically characterized preorder. The converse is not true in general. Therefore our main technical contribution is to provide sufficiently general conditions guaranteeing that consistent and prime formulae are characteristic formulae for some process.

In Sect. 3, we introduce the notion of *decomposable logic* and show that, for such logics, consistent and prime formulae are characteristic for some process (Theorem 2). (Intuitively, a logic is decomposable if, for each formula, the set of processes satisfying it includes the set of processes satisfying a characteristic formula and the logic is sufficiently expressive to witness this inclusion.) We then proceed to identify features that make a logic decomposable, thus paving the way to showing the decomposability of a number of logical formalisms (Sect. 3.1). In particular, we prove that if the set of formulae satisfied by each process can be finitely characterized in a suitable technical sense (see Definition 4), then,

under some mild assumptions, the logic is decomposable (Corollary 2). Moreover, such finitely characterized logics can express the characteristic formula for each process (Proposition 6(ii)).

In order to show the applicability of our general framework, we use it in Sects. 4–5 to show that, for a variety of logical characterizations of process semantics, characteristic formulae are exactly the consistent and prime ones. In particular, this applies to all the semantics in van Glabbeek’s branching-time spectrum. In all these cases, there is a perfect match between the behavioural and logical view of processes: not only do the logics characterize processes up to the chosen notion of behavioural relation, but processes represent all the consistent and prime formulae in the logics.

Proofs of most of the technical results can be found in [1].

2 Process Semantics Defined Logically

We assume that \mathcal{L} is a language interpreted over a non-empty set P , which we refer to as a set of processes. Thus, \mathcal{L} is equipped with a semantic function $\llbracket \cdot \rrbracket_{\mathcal{L}} : \mathcal{L} \rightarrow \mathcal{P}(P)$ (where $\mathcal{P}(P)$ denotes the powerset of P), and we say that $p \in P$ satisfies $\phi \in \mathcal{L}$ whenever $p \in \llbracket \phi \rrbracket_{\mathcal{L}}$. For all $p, q \in P$, we define the following notions:

- $\mathcal{L}(p) = \{\phi \in \mathcal{L} \mid p \in \llbracket \phi \rrbracket_{\mathcal{L}}\}$: the set of formulae in \mathcal{L} that p satisfies; we assume $\mathcal{L}(p) \neq \emptyset$, for each $p \in P$;
- $p^{\uparrow \mathcal{L}} = \{p' \in P \mid \mathcal{L}(p) \subseteq \mathcal{L}(p')\}$: the *upwards closure* of p (with respect to \mathcal{L});
- p and q are *logically equivalent* if $\mathcal{L}(p) = \mathcal{L}(q)$;
- p and q are *incomparable* (with respect to \mathcal{L}) iff neither $\mathcal{L}(p) \subseteq \mathcal{L}(q)$ nor $\mathcal{L}(q) \subseteq \mathcal{L}(p)$ holds.

We say that a formula $\phi \in \mathcal{L}$ is *consistent* iff $\llbracket \phi \rrbracket_{\mathcal{L}} \neq \emptyset$. Formulae $\phi, \psi \in \mathcal{L}$ are said to be *logically equivalent* (or simply *equivalent*) iff $\llbracket \phi \rrbracket_{\mathcal{L}} = \llbracket \psi \rrbracket_{\mathcal{L}}$. When it is clear from the context, we omit the logic \mathcal{L} in the subscript (and in the text). For example, we write $\llbracket \phi \rrbracket$ and p^{\uparrow} instead of $\llbracket \phi \rrbracket_{\mathcal{L}}$ and $p^{\uparrow \mathcal{L}}$. We note that $\mathcal{L}(p) \subseteq \mathcal{L}(q)$ defines a preorder between processes, which we refer to as the *logical preorder* characterized by \mathcal{L} . We say that a preorder over P is *logically characterized* or simply *logical* if it is characterized by some logic \mathcal{L} .

For a subset $S \subseteq P$ we say that:

- S is *upwards closed* iff $p^{\uparrow} \subseteq S$ for all $p \in S$;
- $p \in S$ is *minimal* in S iff for each $q \in S$, if $\mathcal{L}(q) \subseteq \mathcal{L}(p)$ then $\mathcal{L}(q) = \mathcal{L}(p)$;
- $p \in S$ is a *least element* in S iff $\mathcal{L}(p) \subseteq \mathcal{L}(q)$ for each $q \in S$.

Clearly, if p is a least element in a set S , then p is also minimal in S . Notice that, if a set S contains a least element, then it is the unique minimal element in S , up to logical equivalence.

2.1 Characteristic and Prime Formulae

We introduce here the crucial notion of *characteristic formula* for a process [3,10,13] and the one of *prime formula* [2,5], in the setting of logical preorders over processes. Our aim in this study is to investigate when these notions coincide, thus providing a characterization of logically defined processes by means of prime formulae, which sometimes we will refer to as *characterization by primality*. To begin with, in this section we study such a connection in a very general setting. As it turns out, for logically characterized preorders, the property of being characteristic always implies primality (Theorem 1). The main focus of this paper becomes therefore to investigate under what conditions a consistent and prime formula is characteristic for some process in a logical preorder (Sect. 3).

Definition 1 (Characteristic Formula). *A formula $\phi \in \mathcal{L}$ is characteristic for $p \in P$ iff for all $q \in P$ it holds that $q \in \llbracket \phi \rrbracket$ if and only if $\mathcal{L}(p) \subseteq \mathcal{L}(q)$.*

The following simple properties related to characteristic formulae will be useful in what follows.

Proposition 1. *The following properties hold for all $p, q \in P$ and $\phi \in \mathcal{L}$:*

- (i) ϕ is characteristic for p if and only if $\llbracket \phi \rrbracket = p^\dagger$;
- (ii) a characteristic formula for p , if it exists, is unique up to logical equivalence (and can therefore be referred to as $\chi(p)$);
- (iii) if the characteristic formulae for p and q , namely $\chi(p)$ and $\chi(q)$, exist then $\llbracket \chi(p) \rrbracket \subseteq \llbracket \chi(q) \rrbracket$ if and only if $\mathcal{L}(q) \subseteq \mathcal{L}(p)$.

Next we state two useful properties.

Proposition 2. *The following properties hold: (i) for each $\phi \in \mathcal{L}$, $\llbracket \phi \rrbracket$ is upwards closed, and (ii) if $p \in \llbracket \phi \rrbracket \subseteq \llbracket \chi(p) \rrbracket$, then $\llbracket \phi \rrbracket = \llbracket \chi(p) \rrbracket$.*

We now define what it means for a formula to be prime.

Definition 2 (Prime Formula). *We say that $\phi \in \mathcal{L}$ is prime iff for each non-empty, finite subset of formulae $\Psi \subseteq \mathcal{L}$ it holds that $\llbracket \phi \rrbracket \subseteq \bigcup_{\psi \in \Psi} \llbracket \psi \rrbracket$ implies $\llbracket \phi \rrbracket \subseteq \llbracket \psi \rrbracket$ for some $\psi \in \Psi$.*

Observe that our definition is a semantic version of the one given in [5]. This serves our purpose to keep the discussion as abstract as possible. In this perspective, we want to abstract (at least at this point of the discussion) from the syntactic details of the logical formalism, while the classic definition tacitly applies only to languages that feature at least the Boolean connective \vee .

We provide here the first piece of our characterization by primality, by showing that the property of being characteristic implies primality without any extra assumption on the language \mathcal{L} or its interpretation.

Theorem 1. *Let $\phi \in \mathcal{L}$. If ϕ is a characteristic formula for some $p \in P$, then ϕ is prime and consistent.*

Proof. The formula ϕ is obviously consistent because $p \in \llbracket \chi(p) \rrbracket = \llbracket \phi \rrbracket$. Towards proving that $\chi(p)$ is prime, we assume that $\llbracket \chi(p) \rrbracket \subseteq \bigcup_{i \in I} \llbracket \psi_i \rrbracket$, where I is finite and non-empty. By our assumption, since $p \in \llbracket \chi(p) \rrbracket$, then for some $i \in I$, $p \in \llbracket \psi_i \rrbracket$ holds. As, by Proposition 2(i), $\llbracket \psi_i \rrbracket$ is upwards closed, using Proposition 1(i) we can conclude that $\llbracket \chi(p) \rrbracket = p^\uparrow \subseteq \llbracket \psi_i \rrbracket$ as we wanted to prove. \square

Notice that the converse is not true in general, that is, there exist formulae that are consistent and prime but not characteristic. To see this, let $P = \mathbb{Q}$, $\mathcal{L} = \mathbb{R}$ and $\llbracket \phi \rrbracket = \{p \in \mathbb{Q} \mid \phi \leq p\}$. Clearly, all formulae are consistent. Then, $\mathcal{L}(p) = \{\phi \in \mathbb{R} \mid \phi \leq p\}$ which implies that $\mathcal{L}(p) \subseteq \mathcal{L}(q)$ iff $p \leq q$ iff $q \in \llbracket p \rrbracket$. This means that, for each $p \in \mathbb{Q}$, $\phi = p$ is characteristic for p and therefore the characteristic formula is well-defined for all $p \in P$. Furthermore $\llbracket \phi \rrbracket \cup \llbracket \psi \rrbracket = \{p \in \mathbb{Q} \mid \min\{\phi, \psi\} \leq p\}$ for all $\phi, \psi \in \mathcal{L}$, which implies that all formulae are prime. On the other hand $\phi = \sqrt{2} \notin \mathbb{Q}$ cannot be characteristic for any process as $\llbracket \sqrt{2} \rrbracket$ does not have a least element.

3 Characterization by Primality for Logical Preorders

In this section we introduce sufficient conditions under which the converse of Theorem 1 is also true for logical preorders, that is, conditions guaranteeing that every consistent, prime formula is characteristic.

As a first step, we introduce the notion of *decomposable* logic. We show that if a logic is decomposable, then we have a logical characterization of processes by primality. Some of the results involve the Boolean connectives \wedge and \vee , whose intended semantics is the standard one.

Definition 3 (Decomposability). *We say that a formula $\phi \in \mathcal{L}$ is decomposable iff $\llbracket \phi \rrbracket = \llbracket \chi(p) \rrbracket \cup \llbracket \psi_p \rrbracket$ for some $p \in P$ and $\psi_p \in \mathcal{L}$, with $p \notin \llbracket \psi_p \rrbracket$. We say that \mathcal{L} is decomposable iff all consistent formulae $\phi \in \mathcal{L}$ are either decomposable or characteristic for some $p \in P$.*

The following theorem allows us to reduce the problem of relating the notions of prime and characteristic formulae in a given logic to the problem of establishing the decomposability property for that logic. This provides us with a very general setting towards characterization by primality.

Theorem 2. *If \mathcal{L} is decomposable then every formula that is consistent and prime is also characteristic for some $p \in P$.*

3.1 Paths to Decomposability

The aim of this section is to identify features that make a logic decomposable, thus paving the way towards showing the decomposability of a number of logical formalisms in the next sections. First, we observe that if a characteristic formula $\chi(p)$ exists for every $p \in P$, then what we are left to do is to define, for each $\phi \in \mathcal{L}$, a formula ψ_p , for some $p \in P$, with the properties mentioned in Definition 3, as captured by the following proposition.

Proposition 3. *Let \mathcal{L} be a logic such that (i) $\chi(p)$ exists for each $p \in P$, and (ii) for each consistent formula ϕ there exist $p \in \llbracket \phi \rrbracket$ and $\psi_p \in \mathcal{L}$ such that $p \notin \llbracket \psi_p \rrbracket$ and $\llbracket \phi \rrbracket \setminus \llbracket \chi(p) \rrbracket \subseteq \llbracket \psi_p \rrbracket \subseteq \llbracket \phi \rrbracket$. Then \mathcal{L} is decomposable.*

Clearly, when dealing with formalisms featuring at least the Boolean operators \neg and \wedge , as it is the case with the logic for the bisimulation semantics in Sect. 5, such a formula ψ_p is easily defined as $\neg\chi(p) \wedge \phi$. This is stated in the following corollary.

Corollary 1. *Let \mathcal{L} be a logic that features at least the Boolean connective \wedge and such that, for each $p \in P$, the formula $\chi(p)$ exists and there is some formula $\bar{\chi}(p) \in \mathcal{L}$ where $\llbracket \bar{\chi}(p) \rrbracket = \llbracket \chi(p) \rrbracket^c$ (the complement of $\llbracket \chi(p) \rrbracket$). Then \mathcal{L} is decomposable.*

The situation is more complicated when it comes to the other logics for the semantics in the branching-time spectrum (which we consider in Sect. 5) as negation is in general not expressible in these logics, not even for characteristic formulae. Therefore, instead we will prove a slightly stronger statement than the one in Corollary 1 by identifying a weaker condition than the existence of a negation of the characteristic formulae (that we assume to exist) that also leads to decomposability of the logic. This is described in the following proposition.

Proposition 4. *Let $\phi \in \mathcal{L}$, p be a minimal element in $\llbracket \phi \rrbracket$ such that $\chi(p)$ exists in \mathcal{L} , and let $\bar{\chi}(p)$ be a formula in \mathcal{L} such that $\{q \in P \mid \mathcal{L}(q) \not\subseteq \mathcal{L}(p)\} \subseteq \llbracket \bar{\chi}(p) \rrbracket$. Then, $\llbracket \phi \rrbracket \setminus \llbracket \chi(p) \rrbracket \subseteq \llbracket \bar{\chi}(p) \rrbracket$ holds.*

In the next proposition, we build on the above result, and establish some conditions, which are met by the logics we consider in Sect. 5 (apart for the one for bisimulation semantics), and which immediately lead to decomposability.

Proposition 5. *Let \mathcal{L} be a logic that features at least the Boolean connective \wedge and such that:*

- (i) $\chi(p)$ exists for each $p \in P$,
- (ii) for each consistent ϕ , the set $\llbracket \phi \rrbracket$ has a minimal element, and
- (iii) for each $p \in P$, there exists a formula $\bar{\chi}(p)$ such that $p \notin \llbracket \bar{\chi}(p) \rrbracket$ and $\{q \in P \mid \mathcal{L}(q) \not\subseteq \mathcal{L}(p)\} \subseteq \llbracket \bar{\chi}(p) \rrbracket$.

Then, \mathcal{L} is decomposable.

In order to apply the above result to prove decomposability for a logic \mathcal{L} , we now develop a general framework ensuring conditions (i) and (ii) in Proposition 5. To this end, we exhibit a finite characterization of the (possibly) infinite set $\mathcal{L}(p)$ of true facts associated with every $p \in P$. (In order to ensure condition (iii) of the proposition, we will actually construct the formula $\bar{\chi}(p)$ in each of the languages considered in Sect. 5.)

Definition 4 (Characterization). *We say that the logic \mathcal{L} is characterized by a function $\mathcal{B} : P \rightarrow \mathcal{P}(\mathcal{L})$ iff for each $p \in P$, $\mathcal{B}(p) \subseteq \mathcal{L}(p)$ and for each $\phi \in \mathcal{L}(p)$ there exists a non-empty $\Psi \subseteq \mathcal{B}(p)$ such that $\bigcap_{\psi \in \Psi} \llbracket \psi \rrbracket \subseteq \llbracket \phi \rrbracket$. We*

say that \mathcal{L} is finitely characterized by \mathcal{B} iff \mathcal{L} is characterized by a function \mathcal{B} such that $\mathcal{B}(p)$ is finite for each $p \in P$. Finally, we say that \mathcal{B} is monotonic iff $\mathcal{L}(p) \subseteq \mathcal{L}(q)$ implies $\mathcal{B}(p) \subseteq \mathcal{B}(q)$ for all $p, q \in P$.

In what follows, we show that if a logic \mathcal{L} features at least the Boolean connective \wedge and it is finitely characterized by \mathcal{B} , for some monotonic \mathcal{B} , then it fulfils conditions (i) and (ii) in Proposition 5.

Proposition 6. *The following statements hold.*

- (i) *If \mathcal{L} is characterized by \mathcal{B} , then for each $p, q \in P$, $\mathcal{B}(p) \subseteq \mathcal{B}(q)$ implies $\mathcal{L}(p) \subseteq \mathcal{L}(q)$.*
- (ii) *If \mathcal{L} features at least the Boolean connective \wedge and is finitely characterized by \mathcal{B} , then each $p \in P$ has a characteristic formula in \mathcal{L} given by $\chi(p) = \bigwedge_{\phi \in \mathcal{B}(p)} \phi$.*
- (iii) *If \mathcal{L} is finitely characterized by \mathcal{B} , for some monotonic \mathcal{B} , then for each consistent $\phi \in \mathcal{L}$, the set $\llbracket \phi \rrbracket$ has a minimal element.*

It is worth pointing out that the Boolean connective \wedge plays a minor role in (the proof of) Proposition 6(ii). Indeed, it is applied to formulae in $\mathcal{B}(p)$ only. Thus, such a result can be used also to deal with logics that allow for a limited use of such a connective, such as the logics for trace equivalence and other linear-time semantics [9].

Finally, we can summarize the results in this section in the following corollary.

Corollary 2. *Let \mathcal{L} be a logic that features at least the Boolean connective \wedge and such that:*

- (i) *\mathcal{L} is finitely characterized by \mathcal{B} , for some monotonic \mathcal{B} , and*
- (ii) *for each $\chi(p)$, there exists a formula $\bar{\chi}(p)$ such that either*
 - $\llbracket \bar{\chi}(p) \rrbracket = \llbracket \chi(p) \rrbracket^c$, or
 - $p \notin \llbracket \bar{\chi}(p) \rrbracket$ and $\{q \in P \mid \mathcal{L}(q) \not\subseteq \mathcal{L}(p)\} \subseteq \llbracket \bar{\chi}(p) \rrbracket$.

Then, \mathcal{L} is decomposable.

In the remainder of the paper, we will present some applications of our general results.

4 Application to Finitely Many Processes

As a first application, we investigate the case when the set P is finite and the logic \mathcal{L} features at least the Boolean connectives \wedge and \vee . Note that although P itself is finite, it can contain processes with infinite behaviours, e.g., when $p \in P$ represents a labelled transition system with loops. If P is finite, so is \mathcal{L} , up to logical equivalence. Let \mathcal{L}^{fin} be a set of representatives of the equivalence classes of \mathcal{L} modulo logical equivalence, and define $\mathcal{B}^{fin}(p) = \mathcal{L}^{fin}(p) = \mathcal{L}(p) \cap \mathcal{L}^{fin}$, for each $p \in P$. It is easy to see that \mathcal{L} is finitely characterized by \mathcal{B}^{fin} , according

to Definition 4. Moreover, \mathcal{B}^{fin} is clearly monotonic. Thus, by Proposition 6(ii), $\chi(p)$ is well-defined for each p as $\bigwedge_{\psi \in \mathcal{B}^{fin}(p)} \psi$.

In order to show that \mathcal{L} is decomposable, let us consider a consistent formula $\phi \in \mathcal{L}^{fin}$, and let p be minimal in $\llbracket \phi \rrbracket$ (the existence of such a p is guaranteed by the finiteness of P). Now, either $\llbracket \phi \rrbracket = \llbracket \chi(p) \rrbracket$ (in this case we are done), or $\llbracket \phi \rrbracket \setminus \llbracket \chi(p) \rrbracket \neq \emptyset$, and thus, the set $S = \{q \in P \mid q \in \llbracket \phi \rrbracket, \mathcal{L}^{fin}(p) \neq \mathcal{L}^{fin}(q)\}$ is not empty. In this second case it is easy to see that $\psi_p = \bigvee_{q \in S} \chi(q)$ fulfils the requirements of Definition 3. This can be summarized in the following theorem.

Theorem 3 (Characterization by Primality). *Let \mathcal{L} be a logic interpreted over a finite set P that features at least the Boolean connectives \wedge and \vee . Then, each formula $\phi \in \mathcal{L}$ is consistent and prime if and only if ϕ is characteristic for some $p \in P$.*

5 Application to Semantics in van Glabbeek's Spectrum

Our next task is to apply the result described in Corollary 2 to the semantics in the branching-time spectrum, over finite trees and with finite set of actions. All those semantics have been shown to be characterized by specific logics and therefore inherit all the properties of logically defined preorders. We reason about characterization by primality (Theorem 5) by showing that each logic is finitely characterized by some monotonic \mathcal{B} , and by building, for each characteristic formula $\chi(p)$, a formula $\bar{\chi}(p)$ with the properties specified in Proposition 5(iii).

The logics we focus on are the ones for the semantics in *van Glabbeek's branching-time spectrum* [8,9], namely *simulation* (S), *complete simulation* (CS), *ready simulation* (RS), *trace simulation* (TS), *2-nested simulation* (2S), and *bisimulation* (BS). Their syntax and semantics are briefly described in what follows. For a comprehensive overview, we refer the reader to the corresponding literature. In the rest of this section *spectrum* denotes the set {S, CS, RS, TS, 2S, BS} and we let $X \in \text{spectrum}$.

Syntax for Processes. The set of processes P over a finite set of actions Act is given by the following grammar:

$$p ::= 0 \mid ap \mid p + p,$$

where $a \in Act$. Given a process p , we say that p can perform the action a and evolve into p' , denoted $p \xrightarrow{a} p'$, iff (i) $p = ap'$ or (ii) $p = p_1 + p_2$ and either $p_1 \xrightarrow{a} p'$ or $p_2 \xrightarrow{a} p'$ holds. Note that every process denotes a finite loop-free labelled transition system.

We define the set of *initials* of p , denoted $I(p)$, as the set $\{a \in Act \mid p \xrightarrow{a} p' \text{ for some } p' \in P\}$. We write $p \xrightarrow{a}$ if $a \in I(p)$, and we write $p \not\xrightarrow{a}$ if $a \notin I(p)$. We define *traces*(p) as follows:

$$\text{traces}(p) = \{\varepsilon\} \cup \{a\tau \mid \exists a \in Act \exists p' \in P . p \xrightarrow{a} p' \text{ and } \tau \in \text{traces}(p')\}.$$

Table 1. Semantic relations in van Glabbeek’s branching-time spectrum.

Semantic relation	Definition
simulation (S)	$p \lesssim_S q \Leftrightarrow$ for all $p \xrightarrow{a} p'$ there exists $q \xrightarrow{a} q'$ such that $p' \lesssim_S q'$;
complete simulation (CS)	$p \lesssim_{CS} q \Leftrightarrow$ for all $p \xrightarrow{a} p'$ there exists $q \xrightarrow{a} q'$ such that $p' \lesssim_{CS} q'$, and $I(p) = \emptyset$ iff $I(q) = \emptyset$;
ready simulation (RS)	$p \lesssim_{RS} q \Leftrightarrow$ for all $p \xrightarrow{a} p'$ there exists $q \xrightarrow{a} q'$ such that $p' \lesssim_{RS} q'$, and $I(p) = I(q)$;
trace simulation (TS)	$p \lesssim_{TS} q \Leftrightarrow$ for all $p \xrightarrow{a} p'$ there exists $q \xrightarrow{a} q'$ such that $p' \lesssim_{TS} q'$, and $traces(p) = traces(q)$;
2-nested simulation (2S)	$p \lesssim_{2S} q \Leftrightarrow$ for all $p \xrightarrow{a} p'$ there exists $q \xrightarrow{a} q'$ such that $p' \lesssim_{2S} q'$, and $q \lesssim_S p$;
bisimulation (BS)	$p \lesssim_{BS} q \Leftrightarrow$ for all $p \xrightarrow{a} p'$ there exists $q \xrightarrow{a} q'$ such that $p' \lesssim_{BS} q'$, and for all $q \xrightarrow{a} q'$ there exists $p \xrightarrow{a} p'$ such that $p' \lesssim_{BS} q'$

For each trace $\tau = a_1 \dots a_n$, we write $p \xrightarrow{\tau} p'$ for $p \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \dots p_{n-1} \xrightarrow{a_n} p'$. Finally, for each $p \in P$, $dep(p)$ is the length of the longest trace in $traces(p)$.

Behavioural Preorders. The semantics of processes is expressed by preorder relations, which, intuitively, classify processes according to their possible behaviours. Roughly speaking, a process *follows* another in the preorder (or it is *above* it) if it exhibits at least the same behaviours as the latter. The semantic relations in van Glabbeek’s branching-time spectrum are defined as follows.

Definition 5. For each $p, q \in P$ and each $X \in \text{spectrum}$, \lesssim_X is the largest relation satisfying the corresponding condition in Table 1.

It is well-known that $\lesssim_{BS} \subsetneq \lesssim_{2S} \subsetneq \lesssim_{TS} \subsetneq \lesssim_{RS} \subsetneq \lesssim_{CS} \subsetneq \lesssim_S$ [8, 9].

Syntax for Logics. Table 2 provides the definition of the syntax of the logics that capture exactly the above mentioned process semantic relations. We treat formulae of the form $\mathbf{0}$ and $[a]\psi$ as syntactic shorthand for $\bigwedge_{a \in Act} [a]\mathbf{ff}$ and $\neg \langle a \rangle \neg \psi$, respectively. The languages of the different logics yield the following chain of strict inclusions: $\mathcal{L}_S \subsetneq \mathcal{L}_{CS} \subsetneq \mathcal{L}_{RS} \subsetneq \mathcal{L}_{TS} \subsetneq \mathcal{L}_{2S} \subsetneq \mathcal{L}_{BS}$, corresponding to formalisms with strictly increasing expressive power. Notice that, as it will become clear after the definition of the satisfaction relation below, some of the languages present some redundancy, in the sense that they could be replaced with smaller ones, without any loss in expressiveness. For instance, a disjunction is expressible in \mathcal{L}_{BS} using conjunction and negation, and suitably replacing \mathbf{tt} with \mathbf{ff} and vice versa. We followed this approach because we find it helpful to have syntactically larger languages corresponding to more expressive semantics.

Roughly speaking, each language consists of an “existential” and a “universal” sub-language, as highlighted by the definitions in the second and the fourth column of Table 2 ($\phi_X ::= \phi_X^{\exists} \mid \phi_X^{\forall}$ for each $X \in \text{spectrum}$ apart from simulation). The “existential” sub-language (formulae derivable from the non-terminal ϕ_X^{\exists})

is common to all the logics and so is its definition (bottom line of Table 2). The “universal” sub-language (formulae derivable from the non-terminal ϕ_X^\forall) is what actually distinguishes the several languages: its definition is provided for each logic in the corresponding row of Table 2 (see [1] for further explanations and expanded definitions).

Table 2. Syntax of the logics in van Glabbeek’s branching-time spectrum. For every $X \in \{\text{S}, \text{CS}, \text{RS}, \text{TS}, \text{2S}, \text{BS}\}$, the language of \mathcal{L}_X is generated by the grammar rooted in the non-terminal ϕ_X .

Semantics	Syntax	Semantics	Syntax
S	$\phi_S ::= \phi_S^\exists$	TS	$\phi_{\text{TS}} ::= \phi_{\text{TS}}^\exists \mid \phi_{\text{TS}}^\forall$ $\phi_{\text{TS}}^\forall ::= \mathbf{ff} \mid [a]\phi_{\text{TS}}^\forall$
CS	$\phi_{\text{CS}} ::= \phi_{\text{CS}}^\exists \mid \phi_{\text{CS}}^\forall$ $\phi_{\text{CS}}^\forall ::= \mathbf{0}$	2S	$\phi_{\text{2S}} ::= \phi_{\text{2S}}^\exists \mid \phi_{\text{2S}}^\forall$ $\phi_{\text{2S}}^\forall ::= \neg\phi_S$
RS	$\phi_{\text{RS}} ::= \phi_{\text{RS}}^\exists \mid \phi_{\text{RS}}^\forall$ $\phi_{\text{RS}}^\forall ::= [a]\mathbf{ff}$	BS	$\phi_{\text{BS}} ::= \phi_{\text{BS}}^\exists \mid \phi_{\text{BS}}^\forall$ $\phi_{\text{BS}}^\forall ::= \neg\phi_{\text{BS}}$
$\phi_X^\exists ::= \mathbf{tt} \mid \mathbf{ff} \mid \phi_X \wedge \phi_X \mid \phi_X \vee \phi_X \mid \langle a \rangle \phi_X \quad \forall X \in \{\text{S}, \text{CS}, \text{RS}, \text{TS}, \text{2S}, \text{BS}\}$			

Satisfaction Relation. We give here the semantics of the logics, by describing the satisfaction relation for the most expressive one, namely \mathcal{L}_{BS} , corresponding to bisimulation semantics. The semantics for the other logics can be obtained by considering the corresponding subset of clauses.

- $p \in \llbracket \mathbf{tt} \rrbracket$ and $p \notin \llbracket \mathbf{ff} \rrbracket$, for every $p \in P$,
- $p \in \llbracket \phi_1 \wedge \phi_2 \rrbracket$ iff $p \in \llbracket \phi_1 \rrbracket$ and $p \in \llbracket \phi_2 \rrbracket$,
- $p \in \llbracket \phi_1 \vee \phi_2 \rrbracket$ iff $p \in \llbracket \phi_1 \rrbracket$ or $p \in \llbracket \phi_2 \rrbracket$,
- $p \in \llbracket \langle a \rangle \phi \rrbracket$ iff $p' \in \llbracket \phi \rrbracket$ for some $p' \in P$ such that $p \xrightarrow{a} p'$,
- $p \in \llbracket \neg\phi \rrbracket$ iff $p \notin \llbracket \phi \rrbracket$.

The following well-known theorem states the relationship between logics and process semantics that allows us to use our general results about logically characterized semantics.

Theorem 4 (Logical Characterization [8,9]). *For each $X \in \text{spectrum}$ and for all $p, q \in P$, $p \lesssim_X q$ iff $\mathcal{L}_X(p) \subseteq \mathcal{L}_X(q)$.*

We observe that all the logics we consider feature the Boolean connective \wedge , as required by one of the assumptions of Corollary 2. In what follows, we show that every logic meets also the other conditions of the corollary, that is, it is finitely characterized by some monotonic \mathcal{B} , and for each $\chi(p)$ there exists a formula $\bar{\chi}(p)$ such that $p \notin \llbracket \bar{\chi}(p) \rrbracket$ and $\{q \in P \mid \mathcal{L}(q) \not\subseteq \mathcal{L}(p)\} \subseteq \llbracket \bar{\chi}(p) \rrbracket$. We provide here proof details for the illustrative case of *ready simulation* (RS) [4,9,12] only, and refer the interested reader to [1] for further details. We recall the “expanded” syntax of the corresponding logic \mathcal{L}_{RS} :

$$\phi_{\text{RS}} ::= \mathbf{tt} \mid \mathbf{ff} \mid \phi_{\text{RS}} \wedge \phi_{\text{RS}} \mid \phi_{\text{RS}} \vee \phi_{\text{RS}} \mid \langle a \rangle \phi_{\text{RS}} \mid [a]\mathbf{ff}.$$

5.1 Finite Characterization

We prove here that logics in van Glabbeek's branching-time spectrum are finitely characterized by some monotonic \mathcal{B} (condition (i) in Corollary 2).

Lemma 1. *Let $X \in \text{spectrum}$. \mathcal{L}_X is finitely characterized by \mathcal{B} , for some monotonic \mathcal{B} .*

Proof. We detail the case of ready simulation only. For this relation, the function \mathcal{B} is defined as $\mathcal{B}^+(p) \cup \mathcal{B}^-(p)$, where

- $\mathcal{B}^+(p) = \{\mathbf{tt}\} \cup \{\langle a \rangle \varphi \mid \varphi = \bigwedge_{\psi \in \Psi} \psi, \Psi \subseteq \mathcal{B}(p') \text{ and } p \xrightarrow{a} p'\}$, and
- $\mathcal{B}^-(p) = \{[a]\mathbf{ff} \mid p \in \llbracket [a]\mathbf{ff} \rrbracket, a \in \text{Act}\}$.

We have to show that, for each $p \in P$, (i) $\mathcal{B}(p) \subseteq \mathcal{L}(p)$, (ii) $\mathcal{B}(p)$ is finite, (iii) for each $\phi \in \mathcal{L}(p)$ there exists a non-empty Ψ such that $\Psi \subseteq \mathcal{B}(p)$ and $\bigcap_{\psi \in \Psi} \llbracket \psi \rrbracket \subseteq \llbracket \phi \rrbracket$, and (iv) for each $q \in P$, if $\mathcal{L}(p) \subseteq \mathcal{L}(q)$ then $\mathcal{B}(p) \subseteq \mathcal{B}(q)$.

Here we only deal with properties (iii) and (iv). The proof of property (iii) is by induction on the structure of formulae. The cases $\phi = \mathbf{tt}$, $\phi = [a]\mathbf{ff}$, $\phi = \varphi_1 \vee \varphi_2$, and $\phi = \varphi_1 \wedge \varphi_2$ are simple, and are omitted. Assume that $\phi = \langle a \rangle \varphi$. By definition we have that $\varphi \in \mathcal{L}(p')$ for some $p \xrightarrow{a} p'$. By the inductive hypothesis, there exist formulae $\psi_1, \dots, \psi_n \in \mathcal{B}(p')$ (with $n \geq 1$) such that $\bigcap_{i \in \{1, \dots, n\}} \llbracket \psi_i \rrbracket \subseteq \llbracket \varphi \rrbracket$. We define $\psi = \langle a \rangle \bigwedge_i \psi_i$. Clearly, ψ belongs to $\mathcal{B}^+(p)$ (by construction) and $\llbracket \psi \rrbracket \subseteq \llbracket \phi \rrbracket$ (because $\bigcap_{i \in \{1, \dots, n\}} \llbracket \psi_i \rrbracket \subseteq \llbracket \varphi \rrbracket$).

Finally, we show that $\mathcal{B}(p)$ is monotonic (property (iv)). Consider $p, q \in P$, with $\mathcal{L}(p) \subseteq \mathcal{L}(q)$. We want to show that $\phi \in \mathcal{B}(p)$ implies $\phi \in \mathcal{B}(q)$, for each ϕ . Firstly, we observe that, by $\mathcal{L}(p) \subseteq \mathcal{L}(q)$ and Theorem 4, $p \lesssim_{\text{RS}} q$ holds. Thus, we have that $I(p) = I(q)$ and, for each $a \in \text{Act}$ and $p' \in P$ with $p \xrightarrow{a} p'$, there exists $q' \in P$ such that $q \xrightarrow{a} q'$, and $p' \lesssim_{\text{RS}} q'$. Since $I(p) = I(q)$, clearly $\mathcal{B}^-(p) = \mathcal{B}^-(q)$. In order to show that $\mathcal{B}^+(p) \subseteq \mathcal{B}^+(q)$, we proceed by induction on the depth of p . If $I(p) = \emptyset$, then $I(q) = \emptyset$ as well. Thus, we have that $\mathcal{B}^+(p) = \mathcal{B}^+(q) = \{\mathbf{tt}\}$, and the thesis follows. Otherwise ($I(p) \neq \emptyset$), let us consider a formula $\phi = \langle a \rangle \varphi \in \mathcal{B}^+(p)$ (the case when $\psi = \mathbf{tt}$ is trivial). By definition of \mathcal{B}^+ , there exist $p' \in P$, with $p \xrightarrow{a} p'$, and $\Psi \subseteq \mathcal{B}(p')$ such that $\varphi = \bigwedge_{\psi \in \Psi} \psi$. This implies the existence of $q' \in P$ such that $q \xrightarrow{a} q'$ and $p' \lesssim_{\text{RS}} q'$ (and therefore $\mathcal{L}(p') \subseteq \mathcal{L}(q')$). By the inductive hypothesis, $\mathcal{B}(p') \subseteq \mathcal{B}(q')$ holds as well, which means that $\Psi \subseteq \mathcal{B}(q')$. Hence, we have that $\langle a \rangle \varphi \in \mathcal{B}^+(q)$. \square

5.2 Existence of $\bar{\chi}(\cdot)$

In what follows, we show that it is possible to build, for each $\chi(p)$, a formula $\bar{\chi}(p)$, with the properties described in Corollary 2(ii).

Lemma 2. *Let $X \in \text{spectrum}$. For each $\chi(p) \in \mathcal{L}_X$ there exists a formula in \mathcal{L}_X , denoted $\bar{\chi}(p)$, such that (i) $p \notin \llbracket \bar{\chi}(p) \rrbracket$ and (ii) $\{p' \in P \mid p' \not\lesssim p\} \subseteq \llbracket \bar{\chi}(p) \rrbracket$.*

Proof (sketch). We deal with the case of ready simulation only. The formula $\bar{\chi}(p)$ is defined as follows: $\bar{\chi}(p) = \bigvee_{a \notin I(p)} \langle a \rangle \mathbf{tt} \vee \bigvee_{a \in I(p)} [a] \mathbf{ff} \vee \bigvee_{a \in I(p)} \langle a \rangle \bigwedge_{p \xrightarrow{a} p'} \bar{\chi}(p')$. Both properties can be easily proved by induction on the depth of p .

Finally, the following theorem states our main result of this section.

Theorem 5 (Characterization by Primality). *Let $X \in \text{spectrum}$ and let $\phi \in \mathcal{L}_X$. Then, ϕ is consistent and prime if and only if ϕ is characteristic for some $p \in P$.* \square

6 Conclusions

In this paper, we have provided general sufficient conditions guaranteeing that formulae for which model checking can be reduced to equivalence/preorder checking are exactly the consistent and prime ones. We have applied our framework to show that characteristic formulae are exactly the consistent and prime ones when the set of processes is finite, as well as for all the semantics in van Glabbeek’s branching-time spectrum. Our results indicate that the “characterization by primality result” first proved by Boudol and Larsen [5] in the context of the modal logic that characterizes modal refinement over modal transition systems holds in a wide variety of settings in concurrency theory. We feel, therefore, that this study reinforces the view that there is a very close connection between the behavioural and logical view of processes: not only do the logics characterize processes up to the chosen notion of behavioural relation, but processes characterize all the prime and consistent formulae.

In this paper, we have presented applications of our general framework to branching-time semantics. However, ongoing, preliminary investigations indicate that our framework can also be applied to obtain characterization by primality results for the logics for the linear-time semantics in van Glabbeek’s spectrum [9]. By way of example, we mention here that we have already obtained such characterizations for trace, complete trace, failures, readiness, possible futures, and impossible futures semantics. We leave the study of further applications and of possible generalizations of our results for future work.

References

1. Aceto, L., Della Monica, D., Fábregas, I., Ingólfssdóttir, A.: When are prime formulae characteristic? (extended version). <http://www.icetcs.ru.is/dario/techrep/lc15.pdf>
2. Aceto, L., Fábregas, I., de Frutos Escrig, D., Ingólfssdóttir, A., Palomino, M.: Graphical representation of covariant-contravariant modal formulas. In: Proceedings of the 18th EXPRESS. EPTCS, vol. 64, pp. 1–15 (2011)
3. Aceto, L., Ingólfssdóttir, A., Larsen, K.G., Srba, J.: Reactive Systems: Modelling, Specification and Verification. Cambridge University Press, New York (2007)
4. Bloom, B., Istrail, S., Meyer, A.R.: Bisimulation can’t be traced. *J. ACM* **42**(1), 232–268 (1995)

5. Boudol, G., Larsen, K.G.: Graphical versus logical specifications. *Theor. Comput. Sci.* **106**(1), 3–20 (1992)
6. Clarke, E.M., Grumberg, O., Peled, D.A.: *Model Checking*. MIT Press, Cambridge (1999)
7. Cleaveland, R., Steffen, B.: Computing behavioural relations, logically. In: Albert, J.L., Monien, B., Artalejo, M.R. (eds.) *Automata, Languages and Programming*. LNCS, vol. 510, pp. 127–138. Springer, Heidelberg (1991)
8. de Frutos-Escrig, D., Gregorio-Rodríguez, C., Palomino, M., Romero-Hernández, D.: Unifying the linear time-branching time spectrum of process semantics. *Logical Methods Comput. Sci.* **9**(2), 1–74 (2013)
9. van Glabbeek, R.J.: The linear time-branching time spectrum I: The semantics of concrete, sequential processes. In: *Handbook of Process Algebra*, pp. 3–99. Elsevier (2001)
10. Graf, S., Sifakis, J.: A modal characterization of observational congruence on finite terms of CCS. *Inf. Control* **68**(1–3), 125–145 (1986)
11. Keller, R.M.: Formal verification of parallel programs. *Commun. ACM* **19**(7), 371–384 (1976)
12. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. *Inf. Comput.* **94**(1), 1–28 (1991)
13. Steffen, B., Ingólfssdóttir, A.: Characteristic formulae for processes with divergence. *Inf. Comput.* **110**(1), 149–163 (1994)

Stochastization of Weighted Automata

Guy Avni^(✉) and Orna Kupferman

School of Computer Science and Engineering, The Hebrew University,
Jerusalem, Israel
guy.avni@mail.huji.ac.il

Abstract. *Nondeterministic weighted finite automata* (WFAs) map input words to real numbers. Each transition of a WFA is labeled by both a letter from some alphabet and a weight. The weight of a run is the sum of the weights on the transitions it traverses, and the weight of a word is the minimal weight of a run on it. In *probabilistic weighted automata* (PWFAs), the transitions are further labeled by probabilities, and the weight of a word is the expected weight of a run on it. We define and study *stochastization* of WFAs: given a WFA \mathcal{A} , stochastization turns it into a PWFA \mathcal{A}' by labeling its transitions by probabilities. The weight of a word in \mathcal{A}' can only increase with respect to its weight in \mathcal{A} , and we seek stochastizations in which \mathcal{A}' α -approximates \mathcal{A} for the minimal possible factor $\alpha \geq 1$. That is, the weight of every word in \mathcal{A}' is at most α times its weight in \mathcal{A} . We show that stochastization is useful in reasoning about the competitive ratio of randomized online algorithms and in approximated determinization of WFAs. We study the problem of deciding, given a WFA \mathcal{A} and a factor $\alpha \geq 1$, whether there is a stochastization of \mathcal{A} that achieves an α -approximation. We show that the problem is in general undecidable, yet can be solved in PSPACE for a useful class of WFAs.

1 Introduction

A recent development in formal methods for reasoning about reactive systems is an extension of the Boolean setting to a multi-valued one. The multi-valued component may originate from the system, for example when propositions are weighted or when transitions involve costs and rewards [16], and may also originate from rich specification formalisms applied to Boolean systems, for example when asking quantitative questions about the system [7] or when specifying its quality [1]. The interest in multi-valued reasoning has led to growing interest in *nondeterministic weighted finite automata* (WFAs), which map an input word to a value from a semi-ring over a large domain [12, 23].

Many applications of WFAs use the tropical semi-ring $\langle \mathbb{R}^+ \cup \{\infty\}, \min, +, \infty, 0 \rangle$. There, each transition has a *weight*, the weight of a run is the sum of the

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 278410, and from The Israel Science Foundation (grant no 1229/10).

weights of the transitions taken along the run, and the weight of a word is the minimal weight of a run on it. Beyond the applications of WFAs over the tropical semi-ring in quantitative reasoning about systems, they are used also in text, speech, and image processing, where the costs of the WFA are used in order to account for the variability of the data and to rank alternative hypotheses [11, 24].

A different kind of applications of WFAs uses the semi-ring $\langle \mathbb{R}^+ \cup \{\infty\}, +, \times, 0, 1 \rangle$. There, the weight of a run is the product of the weights of the transitions taken along it, and the weight of a word is the sum of the weights of the runs on it. In particular, when the weights on the transitions are in $[0, 1]$ and form a probabilistic transition function (that is, for every state q and letter σ , the sum of the weights of the σ -transitions from q is 1), we obtain a *probabilistic finite automaton* (PFA, for short). In fact, the probabilistic setting goes back to the 60's [25].

The theoretical properties of WFAs are less clean and more challenging than those of their Boolean counterparts. For example, not all WFAs can be determinized [23], and the problem of deciding whether a given WFA has an equivalent deterministic WFA is open. As another example, the containment problem is undecidable for WFAs [21]. The multi-valued setting also leads to new questions about automata and their languages, like approximated determinization [3] or discounting models [13].

By combining the tropical and the probability semi-rings, we obtain a *probabilistic weighted finite automaton* (PWFA, for short). There, each transition has two weights, which we refer to as the *cost* and the *probability*. The weight that the PWFA assigns to a word is then the expected cost of the runs on it. That is, as in the tropical semi-ring, the cost of each run is the sum the costs of the transitions along the run, and as in probabilistic automata, the contribution of each run to the weight of a word depends on both its cost and probability. While PFAs have been extensively studied (e.g., [6]), we are only aware of [20] in which PWFAs were considered.

We introduce and study *stochastization* of WFAs. Given a WFA \mathcal{A} , stochastization turns it into a PWFA \mathcal{A}' by labeling its transitions with probabilities. Recall that in a WFA, the weight of a word is the minimal weight of a run on it. Stochastization of a WFA \mathcal{A} results in a PWFA \mathcal{A}' with the same set of runs, and the weight of a word is the expected cost of these runs. Accordingly, the weight of a word in \mathcal{A}' can only increase with respect to its weight in \mathcal{A} . Hence, we seek stochastizations in which \mathcal{A}' α -*approximates* \mathcal{A} for the minimal possible factor $\alpha \geq 1$. That is, the weight of every word in \mathcal{A}' is at most α times its weight in \mathcal{A} . We note that stochastization has been studied in the Boolean setting in [14], where a PFA is constructed from an NFA.¹ Before describing our contribution, we motivate stochastization further.

¹ Beyond considering the Boolean setting, the work in [14] concerns the ability to instantiate probabilities so that at least one word is accepted with probability arbitrarily close to 1. Thus, the type of questions and motivations are very different from these we study here in the weighted setting.

In [2], the authors describe a framework for using WFAs over the tropical semi-ring in order to reason about *online algorithms*. An online algorithm can be viewed as a reactive system: at each round, the environment issues a request, and the algorithm should process it. The sequence of requests is not known in advance, and the goal of the algorithm is to minimize the overall cost of processing the sequence. Online algorithms for many problems have been extensively studied [8]. The most interesting question about an online algorithm refers to its *competitive ratio*: the worst-case (with respect to all input sequences) ratio between the cost of the algorithm and the cost of an optimal solution – one that may be given by an *offline* algorithm, which knows the input sequence in advance. An online algorithm that achieves a competitive ratio α is said to be α -*competitive*.

The framework in [2] models optimization problems by WFAs, relates the “unbounded look ahead” of the optimal offline algorithm with nondeterminism, and relates the “no look ahead” of online algorithms with determinism. The framework has been broadened to online algorithms with an extended memory or a bounded lookahead, and to a competitive analysis that takes into account assumptions about the environment [3]. An additional useful broadening of the framework would be to consider *randomized* online algorithms, namely ones that may toss coins in order to choose their actions. Indeed, it is well known that many online algorithms that use randomized strategies achieve a better competitive ratio [8]. As we elaborate in Sect. 3.1, this means that rather than pruning the WFA that models an optimization problem to a deterministic one, we consider its stochastization.

Recall that not all WFAs have equivalent or even α -approximating deterministic WFAs. Stochastization is thus useful in finding an approximate solution to problems that are intractable in the nondeterministic setting and are tractable in the probabilistic one. We describe two such applications. One is *reasoning about quantitative properties of probabilistic systems*. In the Boolean setting, while one cannot model check probabilistic systems, typically given by a Markov chain or a Markov decision process, with respect to a specification given by means of a nondeterministic automaton, it is possible to take the product of a probabilistic system with a deterministic or a probabilistic automaton, making model checking easy for them [26]. In the weighted setting, a quantitative specification may be given by a weighted automaton. Here too the product can be defined only with a deterministic or a probabilistic automaton. By stochastizing a WFA specification, we obtain a PWFA (a.k.a. a *rewarded Markov chain* in this context [17]) and can perform approximated model checking. A second application is *approximated determinization*. Existing algorithms for α -determinization [3, 23] handle families of WFAs in which different cycles that can be traversed by runs on the same word cannot have weights that differ in more than an α multiplicative factor (a.k.a. “the α -twins property”). Indeed, cycles as above induce problematic cycles for subset-construction-type determinization constructions. As we show, stochastization can average such cycles, leading to an approximated-determinization construction that successfully α -determinizes WFAs that do not

satisfy the α -twin property and thus could not be α -determinized using existing constructions. Let us note that another candidate application is *weighted language equivalence*, which is undecidable for WFAs but decidable for PWFA [20]. Unfortunately, however, weighted equivalence becomes pointless once approximation enters the picture.

Given a WFA \mathcal{A} and a factor $\alpha \geq 1$, the *approximated stochastization problem* (AS problem, for short) is to decide whether there is a stochastization of \mathcal{A} that α -approximates it. We study the AS problem and show that it is in general undecidable. Special tractable cases include two types of restrictions. First, restrictions on α : we show that when $\alpha = 1$, the problem coincides with determinization by pruning of WFAs, which can be solved in polynomial time. Then, restrictions on the structure of the WFA: we define the class of constant-ambiguous WFAs, namely WFAs whose degree of nondeterminism is a constant, and show that the AS problem for them is in PSPACE. On the other hand, the AS problem is NP-hard already for 7-ambiguous WFAs, namely WFAs that have at most 7 runs on each word. Even more restricted are tree-like WFAs, for which the problem can be solved in polynomial time, and so is the problem of finding a minimal approximation factor α . We show that these restricted classes are still expressive enough to model interesting optimization problems.

Due to the lack of space, some examples and proofs are omitted and can be found in the full version, in the authors' URLs.

2 Preliminaries

A *nondeterministic finite weighted automaton* on finite words (WFA, for short) is a tuple $\mathcal{A} = \langle \Sigma, Q, \Delta, q_0, \tau \rangle$, where Σ is an alphabet, Q is a finite set of states, $\Delta \subseteq Q \times \Sigma \times Q$ is a total transition relation (i.e., for every $q \in Q$ and $\sigma \in \Sigma$, there is at least one state $q' \in Q$ with $\langle q, \sigma, q' \rangle \in \Delta$), $q_0 \in Q$ is an initial state, and $\tau : \Delta \rightarrow \mathbb{R}^+$ is a weight function that maps each transition to a non-negative real value, which is the cost of traversing this transition. If for every $q \in Q$ and $\sigma \in \Sigma$ there is exactly one $q' \in Q$ such that $\langle q, \sigma, q' \rangle \in \Delta$, then \mathcal{A} is a *deterministic WFA* (DWFA, for short). We assume that all states are reachable from the initial state. Consider a transition $t = \langle q, \sigma, q' \rangle \in \Delta$. We use *source*(t), *label*(t), and *target*(t), to refer to q , σ , and q' , respectively. It is sometimes convenient to use a transition function rather than a transition relation. Thus, we use $\delta_{\mathcal{A}} : Q \times \Sigma \rightarrow 2^Q$, where for $q \in Q$ and $\sigma \in \Sigma$, we define $\delta_{\mathcal{A}}(q, \sigma) = \{p \in Q : \langle q, \sigma, p \rangle \in \Delta\}$. When \mathcal{A} is clear from the context we do not state it implicitly.

A *run* of \mathcal{A} on a word $w = w_1 \dots w_n \in \Sigma^*$ is a sequence of transitions $r = r_1, \dots, r_n$ such that *source*(r_1) $\in Q_0$, for $1 \leq i < n$ we have *target*(r_i) = *source*(r_{i+1}), and for $1 \leq i \leq n$ we have *label*(r_i) = w_i . For a word $w \in \Sigma^*$, we denote by *runs*(\mathcal{A}, w) the set of all runs of \mathcal{A} on w . Note that since Δ is total, there is a run of \mathcal{A} on every word in Σ^* , thus $|\text{runs}(\mathcal{A}, w)| \geq 1$, for all

$w \in \Sigma^*$ ². The value of the run, denoted $val(r)$, is the sum of costs of transitions it traverses. That is, $val(r) = \sum_{1 \leq i \leq n} \tau(r_i)$. We denote by $first(r)$ and $last(r)$ the states in which r starts and ends, respectively, thus $start(r) = source(r_1)$ and $last(r) = target(r_n)$. Since \mathcal{A} is nondeterministic, there can be more than one run on each word. We define the value that \mathcal{A} assigns to the word w , denoted $val(\mathcal{A}, w)$, as the value of the minimal-valued run of \mathcal{A} on w . That is, for every $w \in \Sigma^*$, we define $val(\mathcal{A}, w) = \min\{val(r) : r \in runs(\mathcal{A}, w)\}$.

A *probabilistic finite weighted automaton* on finite words (PWFA, for short) is $\mathcal{P} = \langle \Sigma, Q, D, q_0, \tau \rangle$, where Σ, Q, q_0 , and τ are as in WFAs, and $D : Q \times \Sigma \times Q \rightarrow [0, 1]$ is a *probabilistic transition function*. That is, it assigns for each two states $q, p \in Q$ and letter $\sigma \in \Sigma$ the probability of moving from q to p with letter σ . Accordingly, we have $\sum_{p \in Q} D(q, \sigma, p) = 1$, for every $q \in Q$ and $\sigma \in \Sigma$. We sometimes refer to a transition relation $\Delta_D \subseteq Q \times \Sigma \times Q$ induced by D . For two states $q, p \in Q$ and letter $\sigma \in \Sigma$, we have $\Delta_D(q, \sigma, p)$ iff $D(q, \sigma, p) > 0$. Then, $\tau : \Delta_D \rightarrow \mathbb{R}^+$ assigns positive weights to transitions with a positive probability. As in WFAs, we assume that all states are accessible from the initial state by path with a positive probability. Note that if for every $q \in Q$ and $\sigma \in \Sigma$, there is a state $p \in Q$ with $D(q, \sigma, p) = 1$, then \mathcal{P} is a DWFA.

A run $r = r_1, \dots, r_n$ of \mathcal{P} on $w = w_1 \dots w_n \in \Sigma^*$ is a sequence of transitions defined as in WFAs. The probability of r , denoted $\Pr[r]$, is $\prod_{1 \leq i \leq n} D(r_i)$. Similarly to WFAs, for $w \in \Sigma^*$, we denote by $runs(\mathcal{P}, w)$ the set of all runs of \mathcal{P} on w with positive probability. We define $val(\mathcal{P}, w)$ to be the expected value of a run of \mathcal{P} on w , thus $val(\mathcal{P}, w) = \sum_{r \in runs(\mathcal{P}, w)} \Pr[r] \cdot val(r)$.

We say that a WFA \mathcal{A} is k -ambiguous, for $k \in \mathbb{N}$, if k is the minimal number such that for every word $w \in \Sigma^*$, we have $|runs(\mathcal{A}, w)| \leq k$. We say that a WFA \mathcal{A} is *constant-ambiguous* (a CA-WFA, for short) if \mathcal{A} is k -ambiguous from some $k \in \mathbb{N}$. The definitions for PWFAs are similar, thus CA-PWFAs have a bound on the number of possible runs with positive probability.

A *stochastization* of a WFA $\mathcal{A} = \langle \Sigma, Q, \Delta, q_0, \tau \rangle$ is a construction of a PWFA that is obtained from \mathcal{A} by assigning probabilities to its nondeterministic choices. Formally, it is a PWFA $\mathcal{A}^D = \langle \Sigma, Q, D, q_0, \tau \rangle$ obtained from \mathcal{A} such that D is consistent with Δ . Thus, $\Delta_D = \Delta$. Note that since the transition function of \mathcal{A} is total, there is always a stochastization of \mathcal{A} . Note also that if $\delta(q, \sigma)$ is a singleton $\{p\}$, then $D(q, \sigma, p) = 1$.

Recall that in a nondeterministic WFA, the value of a word is the minimal value of a run on it. Stochastization of a WFA \mathcal{A} results in a PWFA \mathcal{A}^D with the same set of runs, and the value of a word is some average of the values of these runs. Accordingly, the value of a word in \mathcal{A}^D can only increase with respect to its value in \mathcal{A} . We would like to find a stochastization with which \mathcal{A}^D *approximates* \mathcal{A}

² A different way to define WFAs would be to designate a set of accepting states. Then, the language of a WFA is the set of words that have an accepting run, and it assigns values to words in its language. Since it is possible to model acceptance by weights, our definition simplifies the setting and all states can be thought of as accepting.

Consider two weighted automata \mathcal{A} and \mathcal{B} , and a factor $\alpha \in \mathbb{R}$ such that $\alpha \geq 1$. We say that \mathcal{B} α -approximates \mathcal{A} if, for every word $w \in \Sigma^*$, we have $\frac{1}{\alpha} \cdot \text{val}(\mathcal{A}, w) \leq \text{val}(\mathcal{B}, w) \leq \alpha \cdot \text{val}(\mathcal{A}, w)$. We denote the latter also by $\frac{1}{\alpha}\mathcal{A} \leq \mathcal{B} \leq \alpha\mathcal{A}$. When $\alpha = 1$, we say that \mathcal{A} and \mathcal{B} are *equivalent*. Note that \mathcal{A} and \mathcal{B} are not necessarily the same type of automata.

A decision problem and an optimization problem naturally arise from this definition:

- *Approximation stochastization* (AS, for short): Given a WFA \mathcal{A} and a factor $\alpha \geq 1$, decide whether there is a distribution function D such that \mathcal{A}^D α -approximates \mathcal{A} , in which case we say that \mathcal{A}^D is an α -stochastization of \mathcal{A} .
- *Optimal approximation stochastization* (OAS, for short): Given a WFA \mathcal{A} , find the minimal $\alpha \geq 1$ such that there is an α -stochastization of \mathcal{A} .

Recall that for every distribution function D , we have that $\mathcal{A} \leq \mathcal{A}^D$. Thus, in both the AS and OAS problems it is sufficient to require $\mathcal{A}^D \leq \alpha \cdot \mathcal{A}$.

Remark 1 [Tightening the Approximation by a Square-root Factor]. For a WFA \mathcal{A} and $\beta \in [0, 1]$, let \mathcal{A}_β be \mathcal{A} with costs multiplied by β . It is easy to see that for all WFAs \mathcal{A} and \mathcal{B} , we have $\frac{1}{\sqrt{\alpha}} \cdot \mathcal{A} \leq \mathcal{B}_{1/\sqrt{\alpha}} \leq \sqrt{\alpha} \cdot \mathcal{A}$ iff $\mathcal{A} \leq \mathcal{B} \leq \alpha \cdot \mathcal{A}$. In particular, taking \mathcal{B} to be \mathcal{A}^D for some distribution function D for \mathcal{A} , we have that \mathcal{A}^D α -approximates \mathcal{A} iff $\mathcal{A}_{1/\sqrt{\alpha}}^D$ $\sqrt{\alpha}$ -approximates \mathcal{A} . It follows that when altering of weights is possible, we can tighten the approximation by a square-root factor. \square

3 Motivation

In Sect. 1, we discussed the application of stochastization in reasoning about quantitative properties of probabilistic systems, reasoning about randomized online algorithms, and approximated determinization. Below we elaborate on the last two.

3.1 A WFA-Based Approach to Reasoning About Online Algorithms

In this section we describe [2]’s WFA-based approach to reasoning about online algorithms and extend it to account for randomized ones. An online algorithm with requests in Σ and actions in A corresponds to a function $g : \Sigma^+ \rightarrow A$ that maps sequences of requests (the history of the interaction so far) to an action to be taken. In general, the algorithm induces an infinite state space, as it may be in different states after processing different input sequences in Σ^* . For a finite set S of configurations, we say that g *uses memory* S , if there is a regular mapping of Σ^* into S such that g behaves in the same manner on identical continuations of words that are mapped to the same configuration.

We model the set of online algorithms that use memory S and solve an optimization problem P with requests in Σ and actions in A , by a WFA $\mathcal{A}_P = \langle \Sigma, S, \Delta, s_0, \tau \rangle$, such that Δ and τ describe transitions between configurations and their costs, and s_0 is an initial configuration. Formally, $\Delta(s, \sigma, s')$ if the set $A' \subseteq A$ of actions that process the request σ from configuration s by updating the configuration to s' is non-empty, in which case $\tau(\langle s, \sigma, s' \rangle)$ is the minimal cost of an action in A' .

An offline algorithm knows the sequence of requests in advance and thus can resolve nondeterminism to obtain a minimal cost. Accordingly, the cost that an offline algorithm with state space S assigns to a sequence of requests $w \in \Sigma^*$ is exactly $\text{val}(\mathcal{A}_P, w)$. On the other hand, an online algorithm is a DWFA \mathcal{A}'_P obtained from \mathcal{A}_P by pruning nondeterministic choices. The competitive ratio of the online algorithm, namely the ratio between its performance and that of the offline algorithm, on the sequence of requests that maximizes this ratio, is then the factor α such that \mathcal{A}'_P α -approximates \mathcal{A}_P . A randomized online algorithm for P that uses state space S can be viewed as a function from S to a probability distribution on A , which induces a probabilistic transition function on top of \mathcal{A}_P . Consequently, we have the following:

Theorem 1. *Consider an online problem P and a set S of configurations. Let \mathcal{A}_P be a WFA with state space S that models online algorithms for P that use memory S . For all $\alpha \geq 1$, there is a randomized online algorithm for P using memory S that achieves competitive ratio α iff \mathcal{A}_P has an α -stochastization.*

Example 1. The Ski-rental Problem. Assume that renting skis costs \$1 per day and buying skis has a one-time cost of \$ M . The online ski-rental problem copes with the fact it is not known in advance how many skiing days are left. Given an input request “skiing continues today”, the online algorithm should decide whether to buy or rent skis. Typically, it is also assumed that renting skis is only allowed for at most $m \geq M$ consecutive days.

The WFA \mathcal{A} induced by the ski-rental problem with parameters M and m is depicted in Fig. 1. Formally, $\mathcal{A} = \langle \{a\}, \{1, \dots, m, q_{own}\}, \Delta, 1, \tau \rangle$, where Δ and τ are described below. A state $1 \leq i < m$ has two outgoing transitions: $\langle i, a, i + 1 \rangle$ with weight 1, corresponds to renting skis at day i , and $\langle i, a, q_{own} \rangle$ with weight M , corresponding to buying skis at day i . Finally, there are transitions $\langle m, a, q_{own} \rangle$ with weight M and $\langle q_{own}, a, q_{own} \rangle$ with weight 0. The optimal deterministic online algorithm is due to [19]; rent skis for $M - 1$ consecutive days, and buy skis on the M -th day, assuming skiing continues. It corresponds to the DWFA obtained by pruning all transitions but $\langle i, a, i + 1 \rangle$, for $1 \leq i < M$, and $\langle M, a, q_{own} \rangle$. This DWFA achieves an optimal approximation factor of $2 - \frac{1}{M}$.

We describe a simple probabilistic algorithm that corresponds to a stochastization of \mathcal{A} that achieves a better bound of $2 - \frac{1.5}{M}$. Intuitively, before skiing starts, toss a coin. If it turns out “heads”, buy skis on the $(M - 1)$ -th day, and if it turns out “tails”, buy on the M -th day. The corresponding distribution function D is depicted in red in Fig. 1. It is not hard to see that the worst case of this stochastization is attained by the word a^M for which we have $\text{val}(\mathcal{A}, a^M) = M$

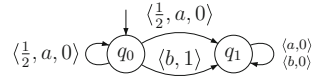
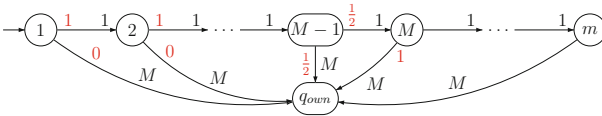


Fig. 1. The WFA that is induced by the ski-rental problem with parameters M and m .

Fig. 2. A PWFA with no equivalent DWFA.

and $val(\mathcal{A}^D, a^M) = \frac{1}{2} \cdot (M - 2 + M) + \frac{1}{2} \cdot (M - 1 + M) = 2M - 1.5$, thus $val(\mathcal{A}^D, a^M) \leq (2 - \frac{1.5}{M}) \cdot val(\mathcal{A}, a^M)$. Finding the optimal distribution function takes care [9, 18] and can achieve an approximation of $1 + \frac{1}{(1+1/M)^{M-1}} \approx e/(e-1) \approx 1.582$ for $M \gg 1$. \square

In the full version, we describe a WFA corresponds to the classical *paging* optimization problem. As we show there, it is sometimes useful to apply the stochastization after extending the state space of \mathcal{A}_P . In the case of paging with a cache of size k , determinization by pruning results in a k -approximation whereas stochastization results in an H_k -approximation, for $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \approx \log(k)$. Thus, allowing the online algorithm to toss coins reduces the competitive ratio from k to H_k .

3.2 Approximated Determinization

Not all WFAs can be determinized. Since some applications require deterministic automata, one way to cope with WFAs that cannot be determinized is to α -determinize them, namely construct a DWFA that α -approximates them, for $\alpha \geq 1$. Our second application is an extension of the class of WFAs that can be approximately determinized.

In [23], Mohri describes a determinization construction for a subclass of WFAs – these that have the *twins property*. In [4], the authors define the α -twins property, for $\alpha \geq 1$, and describe an α -determinization construction for WFAs that satisfy it. We briefly define the properties below. Consider a WFA $\mathcal{A} = \langle \Sigma, Q, \Delta, q_0, \tau \rangle$ and two states $q_1, q_2 \in Q$. We say that q_1 and q_2 are *pairwise reachable* if there is a word $u \in \Sigma^*$ such that there are runs of \mathcal{A} on u that end in q_1 and q_2 . Also, we say that q_1 and q_2 have the t -twins property if they are either not pairwise reachable, or, for every word $v \in \Sigma^*$, if π_1 and π_2 are v -labeled cycle starting from q_1 and q_2 , respectively, then $val(\pi_1) \leq t \cdot val(\pi_2)$. We say that \mathcal{A} has the α -twins property iff every two states in \mathcal{A} have the α -twins property. The α -twins property coincides with Mohri’s twins property when $\alpha = 1$.

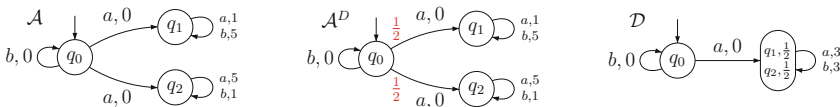


Fig. 3. An illustration of our algorithm for approximated determinization of WFAs.

The α -twins property can be thought of as a *lengthwise* requirement; there might be many runs on a word, but there is a bound on how different the runs are. Our algorithm applies to CA-WFAs. Recall that such WFAs have a dual, *widthwise*, property: the number of runs on a word is bounded by some constant. The algorithm proceeds as follows. Given a CA-WFA \mathcal{A} , we first find an α -stochastization \mathcal{A}^D of it. Since stochastization maintains constant ambiguity, we obtain a CA-PWFA. As we show in Theorem 7, CA-PWFA can be determinized, thus we find an α -determinization of \mathcal{A} .

Example 2. Consider the WFA \mathcal{A} that is depicted in Fig. 3. Note that \mathcal{A} is 2-ambiguous. The optimal stochastization of \mathcal{A} is given by the distribution function D that assigns $D(q_0, a, q_1) = D(q_0, a, q_2) = \frac{1}{2}$. The resulting PWFA \mathcal{A}^D is also depicted in the figure. Then, we construct the DWFA \mathcal{D} by applying the determinization construction of Theorem 7. Clearly, the DWFA \mathcal{D} 3-approximates \mathcal{A} .

We note that \mathcal{A} has the 5-twins property, and this is the minimal t . That is, for every $t < 5$, \mathcal{A} does not have the t -twins property. The DWFA \mathcal{D}' that is constructed from \mathcal{A} using the approximated determinization construction of [4] has the same structure as \mathcal{D} only that the self loops that have weight 3 in \mathcal{D} , have weight 5 in \mathcal{D}' . Thus, \mathcal{D}' 5-approximates \mathcal{A} .

4 Stochastization of General WFAs

In this section we study the AS and OAS problems for general WFA. We start with some good news, showing that the exact stochastization problem can be solved efficiently. Essentially, it follows from the fact that exact stochastization amounts to determinization by pruning, which can be solved in polynomial time [2]. See the full version for details.

Theorem 2. *The exact stochastization problem can be solved in polynomial time.*

We proceed to the bad news.

Theorem 3. *The AS problem is undecidable.*

Proof. In Sect. 1 we mentioned PFA, which add probabilities to finite automata. Formally, a PFA is $\mathcal{P} = \langle \Sigma, Q, P, q_0, F \rangle$, where $F \subseteq Q$ is a set of accepting states and the other components are as in PWFAs. Given a word $w \in \Sigma^*$, each run of \mathcal{P} on w has a probability. The value \mathcal{P} assigns to w is the probability of the accepting runs. We say that \mathcal{P} is *simple* if the image of P is $\{0, 1, \frac{1}{2}\}$. For $\lambda \in [0, 1]$, the λ -emptiness problem for PFAs gets as input a PFA \mathcal{P} , and the goal is to decide whether there is a word $w \in \Sigma^*$ such that $\text{val}(\mathcal{P}, w) > \lambda$. It is well known that the emptiness problem for PFAs is undecidable for $\lambda \in (0, 1)$ [6, 22]. Furthermore, it is shown in [15] that the emptiness problem is undecidable for simple PFAs and $\lambda = \frac{1}{2}$. In the full version we construct, given a simple PFA \mathcal{P} , a WFA \mathcal{A} such that \mathcal{P} is $\frac{1}{2}$ -empty iff there is an $\frac{2+\sqrt{7}}{3}$ -stochastization of \mathcal{A} . \square

5 Stochastization of Constant Ambiguous WFAs

Recall that a CA-WFA has a bound on the number of runs on each word. We show that the AS problem becomes decidable for CA-WFAs. For the upper bound, we show that when the ambiguity is fixed, the problem is in PSPACE. Also, when we further restrict the input to be a *tree-like* WFAs, the OAS problem can be solved in polynomial time. We note that while constant ambiguity is a serious restriction, many optimization problems, including the ski-rental we describe here, induce WFAs that are constant ambiguous. Also, many theoretical challenges for WFAs like the fact that they cannot be determinized, apply already to CA-WFA. We start with a lower bound, which we prove in the full version by a reduction from 3SAT.

Theorem 4. *The AS problem is NP-hard for 7-ambiguous WFAs.*

Consider a k -ambiguous WFA \mathcal{A} , a factor α , and a distribution function D . When k is fixed, it is possible to decide in polynomial time whether \mathcal{A}^D α -approximates \mathcal{A} . Thus, a tempting approach to show that the AS problem is in NP is to bound the size of the optimal distribution function. We show below that this approach is doomed to fail, as the optimal distribution function may involve irrational numbers even when the WFA has only rational weights.

Theorem 5. *There is a 4-ambiguous WFA with rational weights for which every distribution that attains the optimal approximation factor includes irrational numbers.*

Proof. In the full version we construct a WFA \mathcal{A} that includes states q_1, q_2 , and q_3 , where for $i = 1, 2, 3$, the state q_i has nondeterministic choices t_i and t'_i . We define \mathcal{A} so that for $i \neq j \in \{1, 2, 3\}$, an optimal distribution function D satisfies $D(t_i) \cdot D(t_j) = \frac{1}{2}$. Thus, $D(t_i) = \frac{1}{\sqrt{2}}$. \square

We now turn to show that the AS problem is decidable for CA-WFAs. Our proof is based on the following steps: (1) We describe a determinization construction for CA-PWFAs. The probabilities of transitions in the CA-PWFA affects the weights of the transitions in the obtained DWFA. (2) Consider a CA-WFA \mathcal{A} . We attribute each transition of \mathcal{A} by a variable indicating the probability of taking the transition. We define constraints on the variables so that there is a correspondence between assignments and distribution functions. Let \mathcal{A}' be the obtained CA-PWFA. Note that rather than being a standard probabilistic automaton, it is *parameterized*, thus it has the probabilities as variables. Since \mathcal{A}' has the same structure as \mathcal{A} , it is indeed constant ambiguous. (3) We apply the determinization construction to \mathcal{A}' . The variables now appear in the weights of the obtained parameterized DWFA. (4) Given $\alpha \geq 1$, we add constraints on the variables that guarantee that the assignment results in an α -stochastization of \mathcal{A} . For that, we define a parameterized WFA \mathcal{A}'' that corresponds to $\alpha\mathcal{A} - \mathcal{A}'$ and the constraints ensure that it assigns a positive cost to all words. (5) We end up with a system of polynomial constraints, where the system is of size

polynomial in \mathcal{A} . Deciding whether such a system has a solution is known to be in PSPACE.³

We now describe the steps in more detail.

Determinization of CA-WFAs. Before we describe the determinization construction for CA-WFAs, we show that PWFAs are not in general determinizable. This is hardly surprising as neither WFAs nor PFAs are determinizable. We note, however, that the classic examples that show that WFAs are not determinizable use a 2-ambiguous WFA, and as we show below, 2-ambiguous PWFAs are determinizable.

Theorem 6. *There is a PWFA with no equivalent DWFA.*

Proof. Consider the PWFA depicted in Fig. 2. Assume towards contradiction that there is a DWFA \mathcal{D} that is equivalent to \mathcal{A} . Let γ be the smallest absolute value on one of \mathcal{B} 's transitions. Then, \mathcal{B} cannot assign values of higher precision than γ . In particular, for $n \in \mathbb{N}$ such that $2^{-n} < \gamma$, it cannot assign the value 2^{-n} to the word $a^n b$. \square

Consider a PWFA $\mathcal{P} = \langle \Sigma, Q, D, q_0, \tau \rangle$. For a state $q \in Q$ and $\sigma \in \Sigma$, we say that \mathcal{P} has a σ -probabilistic choice at q if there is a transition $\langle q, \sigma, q' \rangle \in \Delta_D$ such that $0 < D(\langle q, \sigma, q' \rangle) < 1$. This is indeed a choice as Δ_D must include a different σ -labeled outgoing transition from q with positive probability. We sometimes refer to such a choice simply as a *probabilistic choice*. We extend the definition to runs as follows. Consider a run $r = r_1, \dots, r_n$ of \mathcal{P} on some word. We say that r makes a probabilistic-choice at index $1 \leq i \leq n$ if there is a $\text{label}(r_i)$ -choice at state $\text{source}(r_i)$. We then say that r *chooses* the transition r_i . Note that when $\Pr[r] > 0$ and the probabilistic choices of r are i_1, \dots, i_ℓ , then $\Pr[r] = \prod_{1 \leq j \leq \ell} D(t_{i_j})$.

Given \mathcal{P} , we construct a DWFA $\mathcal{D} = \langle \Sigma, S, \Delta, q'_0, \tau' \rangle$ equivalent to \mathcal{P} as follows. The states of \mathcal{D} are $S \subseteq 2^{Q \times [0,1]}$. Thus, a state in S is a set of pairs, each consisting of a state q in \mathcal{P} and the probability of reaching q . Thus, the construction is similar to the subset construction used for determinizing NFWs, except that each state in \mathcal{P} is paired with the probability of visiting it in \mathcal{D} . More formally, consider such a state $s = \{\langle q_1, p_1 \rangle, \dots, \langle q_\ell, p_\ell \rangle\}$ and a run r on a word $w \in \Sigma^*$ that ends in s . Then, for $1 \leq i \leq \ell$, we have $p_i = \Pr\{r \in \text{runs}(\mathcal{P}, w) : r \text{ end in } q_i\}$. We define the transitions and their weights accordingly: There is a transition $t = \langle s, \sigma, s' \rangle \in \Delta$ iff for every pair $\langle q'_i, p'_i \rangle \in s'$ we have $p'_i = \sum_{\langle q_j, p_j \rangle \in s} D(q_j, \sigma, q'_i) \cdot p_j$ and $p'_i > 0$. For $s \in S$, the states of s are $st(s) = \{q \in Q : \langle q, p \rangle \in s\}$. The weight of t is $\tau'(t) = \sum_{t' = \langle q_i, \sigma, q'_j \rangle \in st(q) \times \{\sigma\} \times st(q')} \tau(t') \cdot p_j \cdot D(t')$.

While it is not hard to see that \mathcal{D} is equivalent to \mathcal{P} , there is no a-priori bound on the size of \mathcal{D} . In the full version we show that when \mathcal{P} is constant ambiguous,

³ The latter problem, a.k.a. the *existential theory of the reals* [10] is known to be NP-hard and in PSPACE. Improving its upper bound to NP would improve also our bound here.

then \mathcal{D} has a finite number of states. Intuitively, we relate the probabilities that appear in the states of \mathcal{D} with probabilistic choices that the runs of \mathcal{P} perform. Then, we argue that a run of \mathcal{P} on some word $w \in \Sigma^*$ performs at most $k - 1$ probabilistic choices as every choice “spawns” another run of \mathcal{P} on w and $|\text{runs}(\mathcal{P}, w)| \leq k$. Thus, we can bound the number of states in \mathcal{D} , implying the following.

Theorem 7. *Consider a k -ambiguous PWFA \mathcal{P} with m transitions. There is a DWFA \mathcal{D} that is equivalent to \mathcal{P} with $m^{O(k^2)}$ states.*

An Upper Bound for the AS Problem. We are now ready to present the solution to the stochastization problem for CA-WFAs with weights in CQ^+ . Given an input WFA \mathcal{A} and a factor $\alpha \geq 1$, we construct a *polynomial feasibility problem*. The input to such a problem is a set of n variables X and polynomial constraints $P_i(\bar{x}) \leq 0$, for $1 \leq i \leq m$. The goal is to decide whether there is a point $\bar{p} \in \mathbb{R}^n$ that satisfies all the constraints, thus $P_i(\bar{p}) \leq 0$, for $1 \leq i \leq m$. The problem is known to be in PSPACE [10].

Theorem 8. *The AS problem for CA-WFAs is in PSPACE.*

Proof. We describe the intuition and the details can be found in the full version. Consider a k -ambiguous WFA $\mathcal{A} = \langle \Sigma, Q, \Delta, q_0, \tau \rangle$. We associate with \mathcal{A} the following constraints. First, let $X_D = \{x_t : t \in \Delta\}$ be a set of variables, and let \mathcal{A}^D be a parameterized stochastization of \mathcal{A} in which the probability of a transition t is x_t . Next, let \mathcal{D}^D be the parameterized DWFA obtained by applying to \mathcal{A}^D the determinization construction of Theorem 7. Since the variables of \mathcal{A}^D are the probabilities of the transitions and in the determinization construction the probabilities move to the transition weights, the variables in \mathcal{D}^D appear only in the weights.

The first type of constraints are ones that ensure that the assignment to the variables in X_D forms a probabilistic transition function. The second type depends on α and ensures that $\mathcal{D}^D \leq \alpha \cdot \mathcal{A}$. This is done by applying constraints that ensures the emptiness of the parameterized WFA $\mathcal{A}'' = (\alpha \mathcal{A} - \mathcal{D}^D)$. the constraints are based on *Bellman equations*, ensuring that the value of all words in \mathcal{A}'' is positive. The number of additional constraints is then polynomial in n and in the number of transitions of \mathcal{A}'' . Thus, all in all we have polynomially many constraints and the numbers that appear in them are of size polynomial in the size of the weights of \mathcal{A} . Thus, the size of the program is polynomial in the size of \mathcal{A} , and we are done. \square

Remark 2. A different way to specify emptiness by constraints is to consider all simple paths and cycles in the automaton, as was done in [5]. A polynomial treatment of the exponentially many linear constraints then involves a separation oracle, and the ellipsoid method. The method we use here has polynomially many linear constraints to start with, and its implementation is considerably more practical. \square

Remark 3. [Solving the OAS Problem for Tree-like WFAs]. We say that a WFA \mathcal{A} is *tree-like* if the graph induced by its nondeterministic choices is a tree (note that \mathcal{A} is constant ambiguous if the graph is acyclic). Thus, intuitively, for every nondeterministic choice t , there is one way to resolve other nondeterministic choices in order to reach t . Note that the WFA that corresponds to the ski-rental problem as well as the WFA from Example 2 are tree-like. In the full version we show that for every fixed $k \in \mathbb{N}$, the OAS problem can be solved in polynomial time for tree-like k -ambiguous WFAs. \square

References

1. Almagor, S., Boker, U., Kupferman, O.: Formalizing and reasoning about quality. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part II. LNCS, vol. 7966, pp. 15–27. Springer, Heidelberg (2013)
2. Aminof, B., Kupferman, O., Lampert, R.: Reasoning about online algorithms with weighted automata. TALG **6**(2), 1–36 (2010)
3. Aminof, B., Kupferman, O., Lampert, R.: Formal analysis of online algorithms. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 213–227. Springer, Heidelberg (2011)
4. Aminof, B., Kupferman, O., Lampert, R.: Rigorous approximated determinization of weighted automata. TCS **480**, 104–117 (2013)
5. Avni, G., Kupferman, O.: Parameterized weighted containment. TOCL **16**(1), 6 (2014)
6. Azaria, P.: Introduction to Probabilistic Automata. Academic Press, Inc., London (1971)
7. Boker, U., Chatterjee, K., Henzinger, T.A., Kupferman, O.: Temporal specifications with accumulative values. In: Proceedings of the 26th LICS, pp. 43–52 (2011)
8. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
9. Buchbinder, N., Jain, K., Naor, J.S.: Online primal-dual algorithms for maximizing ad-auctions revenue. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 253–264. Springer, Heidelberg (2007)
10. Canny, J.: Some algebraic and geometric computations in PSPACE. In: Proceedings of the STOC, pp. 460–467 (1988)
11. Culik, K., Kari, J.: Digital images and formal languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Beyond Words, pp. 599–616. Springer, Heidelberg (1997)
12. Droste, M., Kuich, W., Vogler, H. (eds.): Handbook of Weighted Automata. Springer, Heidelberg (2009)
13. Droste, M., Rahonis, G.: Weighted automata and weighted logics with discounting. TCS **410**(37), 3481–3494 (2009)
14. Fijalkow, N., Gimbert, H., Horn, F., Oualhadj, Y.: Two recursively inseparable problems for probabilistic automata. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part I. LNCS, vol. 8634, pp. 267–278. Springer, Heidelberg (2014)
15. Gimbert, H., Oualhadj, Y.: Probabilistic automata on finite words: decidable and undecidable problems. In: Abramsky, S., Gavioille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 527–538. Springer, Heidelberg (2010)

16. Henzinger, T.A.: From Boolean to quantitative notions of correctness. In: Proceedings of the 37th POPL, pp. 157–158 (2010)
17. Howard, R.A.: *Dynamic Probabilistic Systems, Volume II: Semi-Markov and Decision Processes*. Vol. 2. Courier Corporation (2013)
18. Karlin, A.R., Manasse, M.S., McGeoch, L.A., Owicki, S.S.: Competitive randomized algorithms for nonuniform problems. *Algorithmica* **11**(6), 542–571 (1994)
19. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoop caching. *Algorithmica* **3**, 77–119 (1988)
20. Kiefer, S., Murawski, A.S., Ouaknine, J., Wachter, B., Worrell, J.: On the complexity of equivalence and minimisation for q -weighted automata. *LMCS*, 9(1) (2013)
21. Krob, D.: The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *IJAC* **4**(3), 405–425 (1994)
22. Madani, O., Hanks, S., Condon, A.: On the undecidability of probabilistic planning and related stochastic optimization problems. *Artif. Intell.* **147**(1–2), 5–34 (2003)
23. Mohri, M.: Finite-state transducers in language and speech processing. *Comput. Linguist.* **23**(2), 269–311 (1997)
24. Mohri, M., Pereira, F.C.N., Riley, M.: Weighted finite-state transducers in speech recognition. *Comput. Speech Lang.* **16**(1), 69–88 (2002)
25. Rabin, M.O.: Probabilistic automata. *Inf. Control* **6**(3), 230–245 (1963)
26. Vardi, M.Y.: Probabilistic linear-time model checking: an overview of the automata-theoretic approach. In: Katoen, J.-P. (ed.) *AMAST-ARTS 1999, ARTS 1999, and AMAST-WS 1999*. LNCS, vol. 1601, p. 265. Springer, Heidelberg (1999)

Algebraic Synchronization Criterion and Computing Reset Words

Mikhail Berlinkov¹(✉) and Marek Szykuła²

¹ Institute of Mathematics and Computer Science,
Ural Federal University, Yekaterinburg, Russia
berlm@mail.ru

² Institute of Computer Science, University of Wrocław, Wrocław, Poland

Abstract. We refine results about relations between Markov chains and synchronizing automata. We express the condition that an automaton is synchronizing in terms of linear algebra, and obtain upper bounds for the reset thresholds of automata with a short word of a small rank. The results are applied to make several improvements in the area.

We improve the best general upper bound for reset thresholds of finite prefix codes (Huffman codes): we show that an n -state synchronizing decoder has a reset word of length at most $O(n \log^3 n)$. Also, we prove the Černý conjecture for n -state automata with a letter of rank at most $\sqrt[3]{6n} - 6$. In another corollary, based on the recent results of Nicaud, we show that the probability that the Černý conjecture does not hold for a random synchronizing binary automaton is exponentially small in terms of the number of states. It follows that the expected value of the reset threshold of an n -state random synchronizing binary automaton is at most $n^{7/4+o(1)}$.

Moreover, reset words of the lengths within our bounds are computable in polynomial time. We present suitable algorithms for this task for various classes of automata for which our results can be applied. These include (quasi-)one-cluster and (quasi-)Eulerian automata.

1 Introduction

We deal with *deterministic finite automata* (DFA) $\mathcal{A} = (Q, \Sigma, \delta)$, where Q is a non-empty *set of states*, Σ is a non-empty *alphabet*, and $\delta: Q \times \Sigma \mapsto Q$ is the complete *transition function*. We extend δ to $Q \times \Sigma^*$ and $2^Q \times \Sigma^*$ as usual, and for the image (resp. preimage) of a set S under a word w we write shortly $S.w$ (resp. $S.w^{-1}$). We denote $\Sigma^{\leq c} = \{w \in \Sigma^* : |w| \leq c\}$, the set of all words over Σ

M. Berlinkov—Supported by the Presidential Program “Leading Scientific Schools of the Russian Federation”, project no. 5161.2014.1, the Russian Foundation for Basic Research, project no. 13-01-00852, the Ministry of Education and Science of the Russian Federation, project no. 1.1999.2014/K, and the Competitiveness Program of Ural Federal University.

M. Szykuła—Supported in part by Polish NCN grant DEC-2013/09/N/-ST6/01194.

of length at most c . The empty word is denoted by ε . Throughout the paper, by n we denote the cardinality $|Q|$, and by k we denote $|\Sigma|$.

A word w *compresses* a subset $S \subseteq Q$ if $|S.w| < |S|$. Then we say that S is *compressible*. The *rank* of a word w is $|Q.w|$. A *reset word* or a *synchronizing word* is a word $w \in \Sigma^*$ of rank 1, that is, w takes the automaton to a particular state no matter of the current state. An automaton is called *synchronizing* if it possesses a reset word. An example of a synchronizing automaton from the Černý series [12] is presented in Fig. 1 (left). One can verify that its shortest reset word is ba^3ba^3b . The length of the shortest reset word is called the *reset threshold* and is denoted by $rt(\mathcal{A})$.

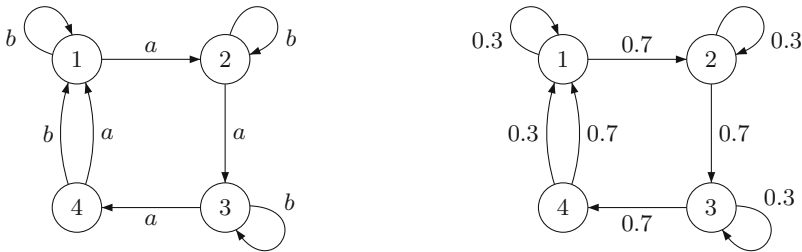


Fig. 1. The automaton \mathcal{C}_4 and the associated Markov chain for $P(a) = 0.7, P(b) = 0.3$

For detailed introduction to the theory of synchronizing automata we refer reader to the surveys [19,27], and for the review of relations with coding theory to [17]. For various applications, reset words allow to reestablish the control under the system modeled by an automaton. So, the reset threshold serves as a natural measure of synchronization. Thus, it is important to compute the reset threshold from both theoretical and practical points of view.

The Černý conjecture, which is arguably the most longstanding open problem in the combinatorial theory of finite automata, states that the reset threshold of a synchronizing automaton is at most $(n - 1)^2$. This bound would be tight, since Černý [12] constructed for each n a synchronizing automaton \mathcal{C}_n with this reset threshold. Moreover, the best upper bound known so far for the reset threshold of a synchronizing n -state automaton is equal to $\frac{n^3-n}{6} - 1$ (for $n \geq 4$) so is cubic in n (see Pin [24]). Thus it is of certain importance to prove specific upper bounds for various classes of synchronizing automata.

In this paper, we improve several results concerning reset thresholds. First, we express the condition that an automaton is synchronizing in terms of linear algebra, and derive upper bounds for automata with a word of a small rank (Sect. 2). Then, we apply the results to improve upper bounds in several cases. In Sect. 3 we show that the Černý conjecture holds for automata with a letter of rank $\sqrt[3]{6n} - 6$, which improves the previous logarithmic result [22]. Also, basing on the recent results of Nicaud [20], we show that the Černý conjecture holds for a random synchronizing binary automaton with probability exponentially (in n) close to 1, and that the expected reset threshold is at most $n^{7/4+o(1)}$.

The next important application of our results is an upper bound for the length of the shortest reset words of finite prefix codes (Huffman codes), which are one of the most popular methods of data compression. One of the problems with compressed data is reliability in case of presence of errors in the compressed text. Eventually, a single error may possibly destroy the whole encoded string. One of the proposed solutions to this problem (for Huffman codes) are codes that can be synchronized by a reset word, regardless of the possible errors. The reset thresholds of binary Huffman codes was first studied by Biskup and Plandowski [8,9], who showed a general upper bound of order $O(n^2 \log n)$, where n is the number of states of the decoder (equivalently, the number of words in the code). They also proved that a word of this length can be computed in polynomial time. The bound was later improved to $O(n^2)$ for a wider class of *one-cluster* automata [2]. In Sect. 4 we prove an upper bound of order $O(n \log^3 n)$. Note that for some applications it can be also important to get bounds in terms of the maximal length of the words in the code (see e.g. [11]).

Unlike the general case, the Černý conjecture has been approved for various classes of automata such as circular [13,23], Eulerian [18] and one-cluster automata with prime length cycle [26]. Later specific quadratic upper bounds for some generalizations of these classes were obtained in [2,5]. However, no efficient algorithm for finding reset words with lengths within the specified bounds has been presented for these classes. Moreover, there is no hope to get a polynomial algorithm for finding the shortest reset words in the general case, since this problem has been shown to be $\text{FP}^{\text{NP}[\log]}$ -hard [21]. Also, unless $\text{P} = \text{NP}$, there is no polynomial algorithm for computing the reset threshold for a given automaton within the approximation ratio n^ε for a certain $\varepsilon > 0$ even in the case of a binary alphabet [15] (cf. also [6,16]).

In Sect. 5 we present polynomial algorithms for finding reset words of length within the proven bounds. Our algorithms can be applied in particular to the classes of decoders of finite prefix codes, and also to generalized classes of quasi-Eulerian and quasi-one-cluster automata. Since from our results it is possible to derive the bounds from [2,5,10,18,25,26], our algorithms apply to these bounds as well.

The full version of this paper is available at [7].

2 Algebraic Synchronization Criterion

In this section we refine some results from [5], formulate the algebraic synchronization criterion, and derive upper bounds for reset thresholds of automata with a word of a small rank. For this purpose, we associate a natural linear structure with an automaton \mathcal{A} . By \mathbb{R}^n we denote the real n -dimensional linear space of row vectors. Without loss of generality, we assume that $Q = \{1, 2, \dots, n\}$ and then assign to each subset $K \subseteq Q$ its *characteristic vector* $[K] \in \mathbb{R}^n$, whose i -th entry is 1 if $i \in K$, and 0, otherwise. For $q \in Q$ we write $[q]$ instead of $[\{q\}]$ to simplify the notation. By $\langle S \rangle$ we denote the linear span of $S \subseteq \mathbb{R}^n$. The $n \times n$ identity matrix is denoted by I_n .

Each word $w \in \Sigma^*$ corresponds to a linear transformation of \mathbb{R}^n . By $[w]$ we denote the matrix of this transformation in the standard basis $[1], \dots, [n]$ of \mathbb{R}^n . For instance, if $\mathcal{A} = \mathcal{C}_4$ from Fig. 1 (left), then

$$[a] = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}, [b] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, [ba] = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

Clearly, the matrix $[w]$ has exactly one non-zero entry in each row. In particular, $[w]$ is *row stochastic*, that is, the sum of entries in each row is equal to 1. In virtue of row-vector notation (apart from [5]), we get that $[uw] = [u][v]$ for every two words $u, v \in \Sigma^*$. By $[w]^T$ we denote the transpose of the matrix $[w]$. One easily verifies that $[S.w^{-1}] = [S][w]^T$. Let us also notice that within this definition the (adjacency) matrix of the underlying digraph of \mathcal{A} is equal to $\sum_{a \in \Sigma} [a]$.

Recall that a word w is a reset word if $q.w^{-1} = Q$, for some state $q \in Q$. Thus, in the language of linear algebra, we can rewrite this fact as $[q][w]^T = [Q]$. For two vectors $g_1, g_2 \in \mathbb{R}^n$, we denote their usual inner (scalar) product by (g_1, g_2) . We say that a vector (matrix) is *positive (non-negative)* if it contains only positive (non-negative) entries. Let $p \in \mathbb{R}_+^n$ be a positive row stochastic vector. Then $([Q], p) = 1$, and a word w is a reset word if and only if there exists $q \in Q$ such that

$$([q.w^{-1}], p) = ([q][w]^T, p) = ([q], p[w]) = 1.$$

Now we need to recall a few properties of Markov chains. A *Markov chain* of an automaton \mathcal{A} is the random walk process of an agent on the underlying digraph of \mathcal{A} where each time an edge labeled by a_i is chosen according to a given probability distribution $P: \Sigma \mapsto \mathbb{R}$. The matrix $S(\mathcal{A}, P) = \sum_{i=1}^k P(a_i)[a_i]$ is called the *transition matrix* of this Markov chain. An example of a Markov chain associated with the automaton $\mathcal{A} = \mathcal{C}_4$ is presented in Fig. 1 (right) for $P(a) = 0.7, P(b) = 0.3$ and its stationary distribution is $\alpha = (\frac{10}{37}, \frac{10}{37}, \frac{10}{37}, \frac{7}{37})$.

A non-negative square matrix M is *primitive* if for some $d > 0$, the matrix M^d is positive. It is well known that if \mathcal{A} is strongly connected and synchronizing, then the matrix of the underlying digraph of \mathcal{A} is primitive, and so is the matrix of a Markov chain of \mathcal{A} for any positive probability distribution P (see [1, 5]). The following proposition is due to the well known Perron-Frobenius theorem.

Proposition 1. *Let M be a row stochastic $n \times n$ matrix. Then there exists a stationary distribution $\alpha \in \mathbb{R}^n$, that is, a non-negative stochastic vector satisfying $\alpha M = \alpha$. Moreover, if M is primitive then α is unique and positive.*

Call a set of words $W \subseteq \Sigma^*$ *complete* for a subspace $V \leq \mathbb{R}^n$, with respect to a vector $g \in V$, if

$$\langle g[w] \mid w \in W \rangle = V.$$

For a subset $S \subseteq Q$ we define $V_S = \langle [p] \mid p \in S \rangle \leq \mathbb{R}^n$.

We aim to strengthen [5, Theorem 9]. Namely, we show that the condition that \mathcal{A} is synchronizing is not necessary if we require completeness for the

corresponding set of words, and that only completeness with respect to the stationary distribution of \mathcal{A} is required. As in [5] we construct an auxiliary automaton. We fix two positive integers d_1, d_2 and two non-empty sets of words $W_1 \subseteq \Sigma^{\leq d_1}$, $W_2 \subseteq \Sigma^{\leq d_2}$. Consider the automaton

$$\mathcal{A}_c(W_1, W_2) = (R, W_2W_1, \delta_{\mathcal{A}_c}),$$

where $R = \{q.w \mid q \in Q, w \in W_1\}$ and $W_2W_1 = \{w_2w_1 \in \Sigma^* \mid w_2 \in W_2, w_1 \in W_1\}$. The transition function $\delta_{\mathcal{A}_c}$ is defined in compliance with the actions of words in \mathcal{A} , i.e. $\delta_{\mathcal{A}_c}(q, w) = \delta(q, w)$, for all $q \in R$ and $w \in W_2W_1$. Note that $\delta_{\mathcal{A}_c}$ is well defined because $q.w \in R$ for all $q \in Q$ and $w \in \Sigma_{\mathcal{A}_c}$. Without loss of generality we may assume that $R = \{1, 2, \dots, r\}$ where $r = |R|$.

Let P_1 and P_2 be some positive probability distributions on the sets W_1 and W_2 , respectively, and denote $[P_i] = \sum_{w \in W_i} P_i(w)[w]$ for $i = 1, 2$. Then the $r \times r$ submatrix formed by the first r rows and the first r columns of the matrix

$$S(\mathcal{A}_c, P_2P_1) = [P_2][P_1] = \sum_{w_1 \in W_1, w_2 \in W_2} P_1(w_1)P_2(w_2)[w_2][w_1]$$

is the transition matrix of the Markov chain on \mathcal{A}_c . By Proposition 1 there exists a steady state distribution $\alpha = \alpha(\mathcal{A}_c) \in V_R$, that is, a stochastic vector (with first r non-negative entries) satisfying $\alpha S(\mathcal{A}_c, P_2P_1) = \alpha$.

For a vector $g \in \mathbb{R}_+^n$, by $DS(g)$ we denote the number of different positive sums of entries of g , i.e. $DS(g) = |\{(g, z) \mid z \in \{0, 1\}^n\}| - 1$.

Theorem 1. *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be an automaton and let*

$$\mathcal{B} = \mathcal{A}_c(W_1, W_2) = (R, W_2W_1, \delta_{\mathcal{B}}),$$

be the automaton defined as above. If W_2W_1 is complete for V_R with respect to α , and $w_0 \in \Sigma^$ is a word with $Q.w_0 = R$, then:*

1. *If $x \in V_R \setminus \langle [R] \rangle$, then there exists $w \in W_2W_1$ such that $(x, \alpha[w]) > (x, \alpha)$;*
2. *\mathcal{B} is synchronizing and $\text{rt}(\mathcal{B}) \leq DS(\alpha) - 1$;*
3. *\mathcal{A} is synchronizing and*

$$\text{rt}(\mathcal{A}) \leq \begin{cases} |w_0| + \text{rt}(\mathcal{B})(d_1 + d_2) \leq |w_0| + (DS(\alpha) - 1)(d_1 + d_2) & \text{if } R \neq Q, \\ 1 + (DS(\alpha) - 2)(d_1 + d_2) & \text{if } R = Q. \end{cases}$$

Proof. Let $x \in V_R \setminus \langle [R] \rangle$. We have

$$(x, [q]) \neq (x, \alpha) \text{ for some } q \in R. \quad (1)$$

Since $[q] \in V_R$ and W_2W_1 is complete for V_R with respect to α , we can represent it as follows:

$$[q] = \sum_{w_1 \in W_1, w_2 \in W_2} \lambda_{w_1, w_2} \alpha[w_2][w_1] \text{ for some } \lambda_{w_1, w_2} \in \mathbb{R}. \quad (2)$$

Multiplying (2) by the vector $[Q]$ we obtain

$$1 = ([q], [Q]) = \sum_{w_1 \in W_1, w_2 \in W_2} \lambda_{w_1, w_2} (\alpha[w_2][w_1], [Q]) = \sum_{w_1 \in W_1, w_2 \in W_2} \lambda_{w_1, w_2}. \quad (3)$$

Multiplying (2) by the vector x we obtain

$$([q], x) = \sum_{w_1 \in W_1, w_2 \in W_2} \lambda_{w_1, w_2} (\alpha[w_2][w_1], x). \quad (4)$$

Arguing by contradiction, suppose $(x, \alpha[u_2][u_1]) = (x, \alpha)$ for every $u_1 \in W_1, u_2 \in W_2$. Then by (3) and (4) we get that $([q], x) = (x, \alpha)$ contradicts (1). Hence $(x, \alpha[u_2][u_1]) \neq (x, \alpha)$, for some $u_1 \in W_1, u_2 \in W_2$.

Since $\alpha[P_2][P_1] = \alpha$, we have either $(x, \alpha[u_2][u_1]) > (x, \alpha)$ or $(x, \alpha[v_2][v_1]) > (x, \alpha)$ for some other $v_1 \in W_1, v_2 \in W_2$. Thus Claim 1 follows.

The proof of Claims 2 and 3 follows from an application of the *greedy extension algorithm* from Sect. 5. \square

Remark 1. If W_2 is complete for \mathbb{R}^n with respect to some vector g , then W_2W_1 is complete for V_R with respect to g .

Criterion 1. *Let α be a stationary distribution of the Markov chain associated with a strongly connected n -state automaton \mathcal{A} by a given positive probability distribution P on the alphabet Σ . Then \mathcal{A} is synchronizing if and only if there exists a set of words W which is complete for \mathbb{R}^n with respect to α .*

Proof. If \mathcal{A} is synchronizing then for each state $q \in Q$ there is a reset word w_q such that $Q.w_q = q$. Hence, $W = \{w_q \mid q \in Q\}$ is complete for \mathbb{R}^n with respect to α , because $\alpha[w_q] = [q]$.

Let us prove the opposite direction. Set

$$W_1 = \{\varepsilon\}, \quad W_2 = \Sigma^{\leq n-1}, \quad \text{and} \quad [P_2] = \frac{1}{n} \sum_{i=0}^{n-1} [P]^i.$$

Then $\alpha[P_2] = \alpha$, and W_2 is complete for \mathbb{R}^n with respect to α . Hence \mathcal{A} is synchronizing by Theorem 1. \square

Now we can provide an upper bound for the reset threshold, if we can find a short word of a small rank.

Theorem 2. *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a synchronizing automaton. Then there is a unique (strongly connected) sink component $\mathcal{S} = (S, \Sigma, \delta)$. Let w be a word and denote $r = |Q.w|$. Let $0 < d < n$ be the smallest positive integer such that $\Sigma^{\leq d}$ is complete for V_S with respect to any stochastic vector $g \in V_S$ and for each $q \in Q$ there is a word $u_q \in \Sigma^{\leq d}$ such that $q.u_q \in S \cap Q.w$. Then*

$$\text{rt}(\mathcal{A}) \leq \begin{cases} (|w| + d) \left(\frac{r^3 - r}{6} \right) - d & \text{if } r \geq 4; \\ |w| + (|w| + d)(r - 1)^2 & \text{if } r \leq 3. \end{cases}$$

Proof. Let $W_1 = \{w\}$, $W_2 = \Sigma^{\leq d}$, $w_0 = w$, and let P_1, P_2 be arbitrary positive distributions on W_1 and W_2 , respectively. We define $\mathcal{B} = \mathcal{A}_c(W_1, W_2)$ as in Theorem 1, and consider its sink component $\mathcal{C} = \mathcal{S}_c(W_1, W_2) = (Q_C, \Sigma, W_2W_1)$. Clearly $Q_C = Q.w \cap S$, and W_2W_1 is complete for $V_{Q_C} \leq V_S$ with respect to any stochastic vector $g \in V_{Q_C}$. By Criterion 1 we obtain that \mathcal{C} is synchronizing. Since for each $q \in Q.w$ there is a word $u_q \in W_2$ and so $w_q \in W_2W_1$ (a letter of \mathcal{B}) which takes q to Q_C , the automaton \mathcal{B} is synchronizing.

Since \mathcal{B} is synchronizing, $|Q.w_0| = r$, and $|u| \leq |w| + d$ for each $u \in W_2W_1$, we have that $\text{rt}(\mathcal{A}) \leq |w| + \text{rt}(\mathcal{B})(|w| + d)$. By Pin's bound for the reset threshold in the general case [24], $\text{rt}(\mathcal{B}) \leq \frac{r^3 - r}{6} - 1$ for $r \geq 4$. \square

3 The Černý Conjecture and Random Automata

Using the new bound, we can extend the class of automata for which the Černý conjecture is proven. In particular, we can improve the result from [22], where the Černý conjecture is proven for automata with a letter of rank at most $1 + \log_2 n$.

Corollary 1. *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a synchronizing automaton. If there is a letter of rank $r \leq \sqrt[3]{6n - 6}$, then \mathcal{A} satisfies the Černý conjecture.*

Another corollary concerns random synchronizing automata. We consider the uniform distribution P_s on all synchronizing binary automata with n states, which is formally defined by $P_s(\mathcal{A}) = P(\mathcal{A})/P_n$, where P is the uniform distribution on all n^{2n} binary automata, and P_n is the probability that a uniformly random binary automaton is synchronizing. It is known that P_n tends to 1 as n goes to infinity [4, 20].

Given an arbitrary small $\varepsilon > 0$ and n large enough, Nicaud [20] proved that with probability at least $1 - O(n^{-1/8+\varepsilon})$ a uniformly random binary automaton has a reset word of length $n^{1+\varepsilon}$. He also proved that with probability at least $1 - O(\exp(n^{-\varepsilon/4}))$, some word of length $n^{3/4+3\varepsilon}(1 + o(1))$ has rank at most $n^{1/4+2\varepsilon}$. Since the probability that a uniformly random binary automaton is synchronizing tends to 1, this also holds with asymptotically at least the same probability for random synchronizing binary automata. The following statement is a straightforward consequence of this result and our Theorem 2.

Corollary 2. *For any $\varepsilon > 0$ and n large enough, with probability at least $1 - O(\exp(n^{-\varepsilon/4}))$, a random n -state synchronizing automaton with at least two letters has a reset word of length at most $n^{7/4+6\varepsilon}(1 + o(1))$, and so satisfies the Černý conjecture. Therefore, the expected value of the reset threshold is at most $n^{7/4+o(1)}$.*

4 Synchronizing Finite Prefix Codes

A finite prefix code (Huffman code) \mathcal{T} is a set of N ($N > 0$) non-empty words $\{w_1, \dots, w_N\}$ from Σ^* , such that no word in \mathcal{T} is a prefix of another word in \mathcal{T} .

A finite prefix code \mathcal{T} is *maximal* if adding any word $w \in \Sigma^*$ to \mathcal{T} does not result in a finite prefix code. We consider only maximal prefix codes. A *reset word* for the code \mathcal{T} is a word w such that for any $u \in \Sigma^*$ the word uw is a sequence of words from \mathcal{T} .

One can easily see that a finite prefix code corresponds naturally to a DFA called the *decoder*, whose states are proper prefixes of words from this code [9]. Formally, for a finite prefix code \mathcal{T} we have the corresponding *decoder* $\mathcal{A}_{\mathcal{T}}$, which is the DFA (Q, Σ, δ) with $Q = \{q_v \mid v \text{ is a proper prefix of a word in } \mathcal{T}\}$, and δ defined as follows:

$$\delta(q_v, a) = \begin{cases} q_{va} & \text{if } va \notin \mathcal{T}; \\ q_\varepsilon & \text{otherwise.} \end{cases}$$

Clearly, a reset word w for a code is a reset word for its decoder, and $Q \cdot w = \{q_\varepsilon\}$. A decoder naturally corresponds to a rooted k -ary tree, thus the number of states $n = (kN - 1)/(k - 1)$, and it does not depend on the length of the words in the code.

In [8, 9] Biskup and Plandowski gave an $O(nh \log n)$ upper bound for the reset thresholds of binary decoders, where h is the maximum length of a word from the code. Since h can be linear in terms of n , this is an $O(n^2 \log n)$ general bound. Later, it was improved to $O(n^2)$ in [2]. However, in the worst case, only decoders with a reset threshold in $\Theta(n)$ are known [9], and it was conjectured that every synchronizing decoder possess a synchronizing word of length $O(n)$. Thus, there was a big gap between the upper and lower bounds for the worst case. The following lemma is a simple generalization of [9, Lemma 14] to k -ary decoders.

Lemma 1. *Let $\mathcal{A}_{\mathcal{T}} = (Q, \Sigma, \delta)$ be the n -state k -ary synchronizing decoder of a finite prefix code \mathcal{T} . There is a word w of rank $r \leq \lceil \log_k n \rceil$ and length r .*

Since there exists a short word of small rank r , we can apply Theorem 2 to improve the general upper bounds for the reset threshold of decoders.

Corollary 3. *Let $\mathcal{A}_{\mathcal{T}} = (Q, \Sigma, \delta)$ be the n -state k -ary synchronizing decoder of a finite prefix code \mathcal{T} , and let $r = \lceil \log_k n \rceil$. Then*

$$\text{rt}(\mathcal{A}_{\mathcal{T}}) \leq \begin{cases} 2 + (r + n - 1) \left(\frac{r^3 - r}{6} - 1 \right) & \text{if } r \geq 4; \\ 2 + (r + n - 1)(r - 1)^2 & \text{if } r \leq 3. \end{cases}$$

If the size k of the alphabet is fixed, Corollary 3 yields $O(n \log^3 n)$ upper bound for the reset threshold, and $O(n \log^2 n)$ upper bound for the length of a word compressing a pair of states of a decoder.

Note that the word w from Lemma 1 can be easily computed in $O(n^2)$ time, since there are $O(n)$ words of length at most $\lceil \log_k n \rceil$. Then a reset word within the bound of Corollary 3 can be computed in polynomial time by the algorithm discussed in Sect. 5.

5 Finding Reset Words of the Bounded Lengths

Throughout this section suppose we are given a strongly connected automaton \mathcal{A} , a word w_0 such that $Q.w_0 = R$ for some $R \subseteq Q$, a non-empty polynomial set of words W_1 with a positive distribution P_1 , and a set of words W_2 with a positive distribution P_2 , which satisfy Theorem 1.

Consider the case when W_2 is of polynomial size. Then we can calculate the dominant eigenvector $\alpha \in \mathbb{R}^n$ of the matrix $[P_2][P_1]$. Under certain assumptions on rationality of the distributions, it can be done in polynomial time. Next, depending on whether the bound is obtained by Theorem 2 or Claim 2 of Theorem 1, we use either a greedy compressing algorithm (such as in [14]), or the following *greedy extension algorithm*, respectively.

The Greedy Extension Algorithm. We start from $x_0 = [q]$ for $q \in R$ and by Claim 1 of Theorem 1 find $u_0 \in W_2W_1$ such that $(x_0, \alpha[u_0]) > (x_0, \alpha)$. For $i = 0, 1, \dots$ following this way until $x_i \in \langle [R] \rangle$, find for $x_{i+1} = x_i[u_i]^t$ a word $u_{i+1} \in W_2W_1$ such that $(x_{i+1}, \alpha[u_{i+1}]) > (x_{i+1}, \alpha)$. Since x_i is a 1-0 vector, we need at most $\text{DS}(\alpha) - 1$ steps until $x_i = [q]([u_i u_{i-1} \dots u_0])^t = [R]$. As the result we return the word $w_0 u_i u_{i-1} \dots u_0$. Notice that in the case when $R = Q$ we can choose q such that for some letter $a \in \Sigma$, we have $|q.a^{-1}| > 1$ and set $u_0 = a$. \square

The problem is that usually W_2 is given by $\Sigma^{\leq d}$ for some $d = \text{poly}(n)$. The following reduction procedure allows to replace potentially exponential set W_2 with a polynomial set of words W , whose the longest words are not longer than those of W_2 .

The Reduction Procedure. The procedure takes a number $d \geq 0$, and returns a polynomial subset $W \subseteq \Sigma^{\leq d}$ such that $\langle W \rangle = \langle \Sigma^{\leq d} \rangle$ and the maximum length of words from W is the shortest possible.

We start with $V_0 = \{I_n\}$ and $W = \{\varepsilon\}$. In each iteration $i \in \{1, 2, \dots\}$ we first set $V_{i+1} = V_i$. Then we subsequently check each letter $a \in \Sigma$ and each word $u \in W$ of length i : If the matrix $[ua]$ does not belong to the subspace V_{i+1} , we add the word ua to W and the matrix $[ua]$ to the basis of V_{i+1} . We stop the procedure at the first iteration where nothing is added.

Since in an i -th iteration we have considered $a \in \Sigma$ and $u \in W$ of length less than i in the previous iterations, by induction we get

$$V_i = \langle I_n(W \cap \Sigma^{\leq i}) \rangle = \langle I_n \Sigma^{\leq i} \rangle.$$

It follows from the ascending chain argument (see e.g. [18,26]) that for some $j < n$ we have

$$V_j = V_{j+1} = \dots$$

Thus the procedure is stopped at the first such j , and $j \leq \min\{d, n - 1\}$. We get that $\langle W \rangle = V_j = \langle \Sigma^d \rangle$. Since in each step we add only independent matrices as the basis of V_{i+1} , we get $|W| = \dim(V_j)$. Also the lengths of words in W are at most $j \leq \min\{d, n - 1\}$. \square

Using the reduction procedure for total completeness we can replace Σ^d from Theorem 2 by a polynomial W which is also complete for V_S with respect to any stochastic vector $g \in V_S$. Hence, this yields a polynomial time algorithm finding reset words of lengths within the bound of Theorem 2.

In some situations we are interested only in completeness with respect to a given vector α . Then we can find a reduced set W of potentially shorter words than that obtained by the general reduction procedure.

The Reduction Procedure for α -Completeness. The procedure takes a number $d \geq 0$ and a vector $\alpha \in \mathbb{R}^n$, and returns a polynomial subset $W \subseteq \Sigma^{\leq d}$ such that $\langle \alpha W \rangle = \langle \alpha \Sigma^{\leq d} \rangle$ and the maximum length of words from W is the shortest possible.

We just follow the general reduction procedure, where instead of matrix spaces we consider vector spaces. It is enough to replace I_0 by α , and we obtain $\langle \alpha W \rangle = V_j = \langle \alpha \Sigma^{\leq d} \rangle$. \square

Remark 2. Instead of $\Sigma^{\leq d}$ the reduction procedures can also reduce any set of words $W' \subset \Sigma^*$ that is factor-closed. A set of words W' is *factor-closed* if $uvw \in W'$ implies that $uw \in W'$, for each $u, v, w \in \Sigma^*$.

5.1 Synchronizing Quasi-Eulerian Automata

Let α be the probability distribution on $\Sigma^{\leq d}$ induced by a probability distribution $P: \Sigma \mapsto \mathbb{R}^+$ on the alphabet, that is, $[P_2] = \frac{1}{d+1} \sum_{i=0}^d [P]^i$. Suppose that $d < \text{poly}(n)$ is such that $\Sigma^{\leq d}$ is complete for \mathbb{R}^n with respect to α . Using the reduction procedure, we can construct a set U of at most n words such that $\langle \alpha U \rangle = \langle \alpha \Sigma^{\leq d} \rangle = \mathbb{R}^n$. However, α is not necessarily the stationary distribution for some positive probability distribution on U . The following lemma solves this problem.

Lemma 2. *Let $W = \{au \mid u \in \text{Suff}(U), a \in \Sigma\}$, where $\text{Suff}(U)$ is the set of proper suffixes of U . Then there exists a positive probability distribution on W such that α is the corresponding stationary distribution.*

As an application we get a polynomial algorithm for finding a reset word for the class of *quasi-Eulerian* automata, a generalization of Eulerian automata. We call an automaton \mathcal{A} *quasi-Eulerian* with respect to an integer $c \geq 0$ if it satisfies the following two conditions:

1. there is a subset $E_c \subseteq Q$ containing $n - c$ states such that only one of these states, say s , can have incoming edges from the set $Q \setminus E_c$;
2. there exists a positive probability distribution P on Σ such that the columns of the matrix $[P]$ that correspond to the states from $E_c \setminus \{s\}$ sum up to 1.

Within this definition, for $c = 0$ we get so-called *pseudo-Eulerian* automata, and if additionally P is uniform on Σ , then we get Eulerian automata. The upper bound $1 + (n - 2)(n - 1)$ on the reset thresholds of Eulerian automata was

found by Kari [18], and extended to the class of pseudo-Eulerian automata by Steinberg [25]. These results were generalized in [5, Corollary 11] by showing the upper bound $2^c(n - c + 1)(n - 1)$ for the class of quasi-Eulerian automata with respect to a non-negative integer c . The following theorem gives a polynomial time algorithm for finding reset words satisfying these bounds.

Theorem 3. *Given a synchronizing automaton \mathcal{A} which is quasi-Eulerian with respect to an integer $c \geq 0$, there is a polynomial time algorithm for finding a reset word of length at most:*

$$\begin{cases} 2^c(n - c + 1)d & \text{if } c > 0; \\ 1 + (n - 2)d & \text{if } c = 0, \end{cases}$$

where $d \leq n - 1$ is the smallest integer such that $\Sigma^{\leq d}$ is complete.

5.2 Synchronizing Quasi-One-Cluster Automata

The *underlying digraph* of a letter $a \in \Sigma$ is the digraph with edges labeled by a . Every connected component, called *cluster*, in the underlying digraph of a letter has exactly one cycle, and possible some trees rooted on this cycle. An automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is called *one-cluster* if there is a letter $a \in \Sigma$ whose underlying digraph has only one cluster. An automaton \mathcal{A} is *quasi-one-cluster* with respect to an integer $c \geq 0$ if it has a letter whose underlying digraph has a cluster such that there are at most c states in the cycles of all other clusters. Clearly, one-cluster automata are quasi-one-cluster with respect to $c = 0$.

The Černý conjecture was proved for one-cluster automata with prime length cycle [26]. Also, quadratic bounds for the reset thresholds in the general case of one-cluster automata were presented [2, 3, 10, 25]. In [5] the upper bound $2^c(2n - c - 2)(n - c + 1)$ was proved for quasi-one-cluster with respect to c .

The following theorem gives a polynomial algorithm finding a reset word for quasi-one-cluster automata, whose length is of the mentioned bounds. It can be also easily modified to deal with the bounds from [10] for one-cluster automata.

Theorem 4. *Let \mathcal{A} be a synchronizing automaton that is quasi-one-cluster with respect to a letter a and $c \geq 0$. Let C be the largest cycle of a and h be the maximal height of the trees labeled by a . Let $W_1 = \{a^{h+i} \mid i \in \{0, \dots, |C| - 1\}\}$. Then there is a polynomial algorithm for finding a reset word for \mathcal{A} of length at most*

$$\begin{cases} 2^c(2n - c)(n - c + 1) & \text{if } c > 0; \\ 1 + (2n - r)(n - 2) & \text{if } c = 0, \end{cases}$$

where r is the smallest dimension of $\langle W_1\beta \rangle$ for $\beta \in V_C \setminus \langle [C] \rangle$. In particular, if $|C|$ is prime then $r = |C|$.

References

1. Ananichev, D., Gusev, V., Volkov, M.: Slowly synchronizing automata and digraphs. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 55–65. Springer, Heidelberg (2010)
2. Béal, M.P., Berlinkov, M.V., Perrin, D.: A quadratic upper bound on the size of a synchronizing word in one-cluster automata. *Int. J. Found. Comput. Sci.* **22**(2), 277–288 (2011)
3. Béal, M.-P., Perrin, D.: A quadratic upper bound on the size of a synchronizing word in one-cluster automata. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 81–90. Springer, Heidelberg (2009)
4. Berlinkov, M.V.: On the probability to be synchronizable (2013). <http://arxiv.org/abs/1304.5774>
5. Berlinkov, M.V.: Synchronizing quasi-eulerian and quasi-one-cluster automata. *Int. J. Found. Comput. Sci.* **24**(6), 729–745 (2013)
6. Berlinkov, M.V.: On two algorithmic problems about synchronizing automata. In: Shur, A.M., Volkov, M.V. (eds.) DLT 2014. LNCS, vol. 8633, pp. 61–67. Springer, Heidelberg (2014)
7. Berlinkov, M.V., Szykuła, M.: Algebraic synchronization criterion and computing reset words (2014). <http://arxiv.org/abs/1412.8363>
8. Biskup, M.T.: Shortest synchronizing strings for huffman codes. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 120–131. Springer, Heidelberg (2008)
9. Biskup, M.T., Plandowski, W.: Shortest synchronizing strings for huffman codes. *Theoret. Comput. Sci.* **410**(38–40), 3925–3941 (2009)
10. Carpi, A., D’Alessandro, F.: Independent sets of words and the synchronization problem. *Adv. Appl. Math.* **50**(3), 339–355 (2013)
11. Carpi, A., D’Alessandro, F.: Černý-like problems for finite sets of words. In: Proceedings of the 15th Italian Conference on Theoretical Computer Science, Perugia, Italy, September 17–19, 2014. pp. 81–92 (2014)
12. Černý, J.: Poznámka k homogénnym experimentom s konečnými automatami. *Matematicko-fyzikálny Časopis Slovenskej Akadémie Vied* **14**(3), 208–216 (1964)
13. Dubuc, L.: Sur les automates circulaires et la conjecture de Černý. *Informatique Théorique et Applications* **32**, 21–34 (1998)
14. Eppstein, D.: Reset sequences for monotonic automata. *SIAM J. Comput.* **19**, 500–510 (1990)
15. Gawrychowski, P., Straszak, D.: Strong inapproximability of the shortest reset word. In: Italiano, G.F., et al. (eds.) MFCS 2015, Part I, LNCS 9234, pp. 243–255. Springer, Heidelberg (2015)
16. Gerbush, M., Heeringa, B.: Approximating minimum reset sequences. In: Domaratzki, M., Salomaa, K. (eds.) CIAA 2010. LNCS, vol. 6482, pp. 154–162. Springer, Heidelberg (2011)
17. Jürgensen, H.: Synchronization. *Inform. Comput.* **206**(9–10), 1033–1044 (2008)
18. Kari, J.: Synchronizing finite automata on Eulerian digraphs. *Theoret. Comput. Sci.* **295**(1–3), 223–232 (2003)
19. Kari, J., Volkov, M.V.: Černý’s conjecture and the road coloring problem. In: Handbook of Automata. European Science Foundation (to appear)
20. Nicaud, C.: Fast synchronization of random automata (2014). <http://arxiv.org/abs/1404.6962>

21. Olschewski, J., Ummels, M.: The complexity of finding reset words in finite automata. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 568–579. Springer, Heidelberg (2010)
22. Pin, J.E.: Utilisation de l'algèbre linéaire en théorie des automates. In: Act. Colloq. AFCET-SMF Math. Appl. II. pp. 85–92. AFCET (1978)
23. Pin, J.E.: Sur un cas particulier de la conjecture de Černý. Automata, Languages and Programming. LNCS, pp. 345–352. Springer, Heidelberg (1978). in French
24. Pin, J.E.: On two combinatorial problems arising from automata theory. In: Proceedings of the International Colloquium on Graph Theory and Combinatorics, vol. 75, pp. 535–548. North-Holland Mathematics Studies (1983)
25. Steinberg, B.: The averaging trick and the Černý conjecture. Int. J. Found. Comput. Sci. **22**(7), 1697–1706 (2011)
26. Steinberg, B.: The Černý conjecture for one-cluster automata with prime length cycle. Theoret. Comput. Sci. **412**(39), 5487–5491 (2011)
27. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)

Recurrence Function on Sturmian Words: A Probabilistic Study

Valérie Berthé¹, Eda Cesaratto², Pablo Rotondo³,
Brigitte Vallée⁴(✉), and Alfredo Viola³

¹ LIAFA, CNRS UMR 7089, University Paris Diderot, Paris, France
`berthe@liafa.univ-paris-diderot.fr`

² Conicet and Universidad Nacional de General Sarmiento, Buenos Aires, Argentina
`ecesarat@ungs.edu.ar`

³ Universidad de la República, Montevideo, Uruguay
`pabloedrot@gmail.com`, `viola@fing.edu.uy`

⁴ GREYC, CNRS UMR 6072, University de Caen, Caen, France
`brigitte.vallee@unicaen.fr`

Abstract. This paper is a first attempt to describe the probabilistic behaviour of a random Sturmian word. It performs the probabilistic analysis of the recurrence function which provides precise information on the structure of such a word. With each Sturmian word of slope α , we associate particular sequences of factor lengths which have a given “position” with respect to the sequence of continuants of α , we then let α to be uniformly drawn inside the unit interval $[0,1]$. This probabilistic model is well-adapted to better understand the role of the position in the recurrence properties.

1 Introduction

The recurrence function measures the “complexity” of an infinite word and describes the possible occurrences of finite factors inside it together with the maximal gaps between successive occurrences. This recurrence function is thus widely studied, notably in the case of Sturmian words (see [3,9]) which are in a precise sense the simplest infinite words which are not eventually periodic (see e.g. [8]). With each Sturmian word is associated an irrational number α , and many of its characteristics depend on the continued fraction expansion of α . This is in particular the case for the recurrence function $n \mapsto R_\alpha(n)$, where the integer $R_\alpha(n)$ is the length of the smallest “window” which is needed for discovering the set $\mathcal{L}_\alpha(n)$ of all the finite factors of length n inside α . As this set $\mathcal{L}_\alpha(n)$ is widely used in many applications of Sturmian words (for instance quasicrystals, or digital geometry), the function $n \mapsto R_\alpha(n)$ thus intervenes very often as a pre-computation cost, and it is important to better understand this function “on average”, when the real α is randomly chosen in the unit interval.

Most of the classical studies on the recurrence function deal with a *fixed* α , and the usual focus is put on *extremal* behaviours of the recurrence function.

Here, we adopt a “dual” approach which is probabilistic: with each α , we associate *particular* sequences of indices n (i.e., factor lengths), which have a *given* “position” with respect to the sequence of continuants $(q_k(\alpha))_k$, we then let α be *uniformly drawn* inside the unit interval, and we perform a *probabilistic* study to better understand the role of the position in the recurrence function.

The expression of the recurrence function is recalled in Sect. 2. Our viewpoint and our main results are given in Sect. 3. Proofs are provided in Sect. 4.

2 The Recurrence Function of Sturmian Words

Notation. In the sequel $\varphi = (\sqrt{5} - 1)/2 = 0.6180339\dots$ stands for the inverse of the golden ratio, and for two integers a, b , the set of integers n that satisfy $a \leq n \leq b$ is denoted by $\llbracket a, b \rrbracket := [a, b] \cap \mathbb{N}$.

We consider a finite set \mathcal{A} of *symbols*, called *alphabet*. Let $u = (u_n)_{n \in \mathbb{N}}$ be an infinite word in $\mathcal{A}^{\mathbb{N}}$. A finite word w of length n is a factor of u if there exists an index m for which $w = u_m \dots u_{m+n-1}$. Let $\mathcal{L}_u(n)$ stand for the set of factors of length n of u . Two functions describe the set $\mathcal{L}_u(n)$ inside the word u , namely the complexity and the recurrence function.

The (*factor*) *complexity function* of the infinite word u is defined as the sequence $n \mapsto p_u(n) := |\mathcal{L}_u(n)|$. The eventually periodic words are the simplest ones, in terms of the complexity function, and satisfy $p_u(n) \leq n$ for some n . The simplest words that are not eventually periodic satisfy the equality $p_u(n) = n + 1$ for each $n \geq 0$. Such words do exist, they are called *Sturmian words*. Moreover, Morse and Hedlund provided a powerful arithmetic description of Sturmian words (see also [8] for more on Sturmian words).

Proposition 1 (Morse and Hedlund [9]). Associate with a pair $(\alpha, \beta) \in [0, 1]^2$ the two infinite words $\underline{\mathfrak{S}}(\alpha, \beta)$ and $\overline{\mathfrak{S}}(\alpha, \beta)$ whose n -th symbols are respectively

$$\underline{u}_n = \lfloor \alpha(n + 1) + \beta \rfloor - \lfloor \alpha n + \beta \rfloor, \quad \overline{u}_n = \lceil \alpha(n + 1) + \beta \rceil - \lceil \alpha n + \beta \rceil.$$

Then a word $u \in \{0, 1\}^{\mathbb{N}}$ is Sturmian if and only if it equals $\underline{\mathfrak{S}}(\alpha, \beta)$ or $\overline{\mathfrak{S}}(\alpha, \beta)$ for a pair (α, β) formed with an irrational $\alpha \in]0, 1[$ and a real $\beta \in [0, 1[$.

It is also important to study where finite factors occur inside the infinite word u . An infinite word $u \in \mathcal{A}^{\mathbb{N}}$ is *uniformly recurrent* if every factor of u appears infinitely often and with bounded gaps. More precisely, denote by $w_u(q, n)$ the minimal number of symbols u_k with $k \geq q$ which have to be inspected for discovering the whole set $\mathcal{L}_u(n)$ from the index q . Then, the integer $w_u(q, n)$ is a sort of “waiting time”. Then u is uniformly recurrent if each set $\{w_u(q, n); q \in \mathbb{N}\}$ is bounded, and the *recurrence function* $n \mapsto R_u(n)$ is defined as

$$R_u(n) := \max\{w_u(q, n); q \in \mathbb{N}\}.$$

We then recover the usual definition: Any factor of length $R_u(n)$ of u contains all the factors of length n of u , and the length $R_u(n)$ is the smallest integer which satisfies this property. The inequality $R_u(n) \geq p_u(n) + n - 1$ thus holds.

Any Sturmian word is uniformly recurrent. Its recurrence function only depends on the slope α and is thus denoted by $n \mapsto R_\alpha(n)$. Moreover, it only depends on α via its *continuants*. We now recall this notion which plays a central role in the paper. Consider the *continued fraction expansion* of the irrational α

$$\alpha = \frac{1}{m_1 + \frac{1}{\ddots + \frac{1}{m_k + \frac{1}{\ddots}}}} = [m_1, m_2, \dots, m_k, \dots].$$

The positive integers m_k are called the *partial quotients*. The truncated expansion $[m_1, \dots, m_k]$ at depth k defines a rational, and the *continuant* $q_k(\alpha)$ is the denominator of this rational. The continuant sequence satisfies $q_{-1} = 0, q_0 = 1$ and for any $k \geq 1$ the recurrence $q_k = m_k q_{k-1} + q_{k-2}$ for all k .

The following result due to Morse and Hedlund relates the recurrence function $R_\alpha(n)$ and the sequence $k \mapsto q_k(\alpha)$ (Fig. 1).

Proposition 2 (Morse and Hedlund [9]). *For any Sturmian word of slope α , the recurrence function $n \mapsto R_\alpha(n)$ is piecewise affine and satisfies*

$$R_\alpha(n) = n - 1 + q_k(\alpha) + q_{k-1}(\alpha), \text{ for any } n \in \llbracket q_{k-1}(\alpha), q_k(\alpha) - 1 \rrbracket.$$

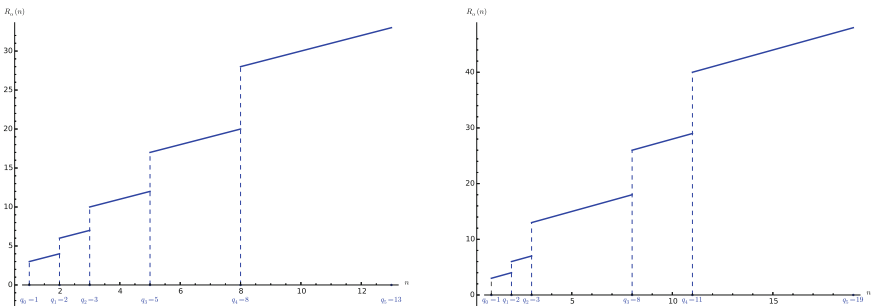


Fig. 1. Two instances of recurrence functions $n \mapsto R_\alpha(n)$ associated with $\alpha = \varphi^2$ (left) and $\alpha = 1/e$ (right), with $\varphi = (\sqrt{5} - 1)/2$ being the inverse of the golden ratio.

It is thus natural to study the quotient $S(\alpha, n) := (R_\alpha(n) + 1)/n$. When n belongs to the interval $\llbracket q_{k-1}(\alpha), q_k(\alpha) - 1 \rrbracket$, this quotient depends itself on two quotients: the quotient $x_k(\alpha) := q_{k-1}(\alpha)/q_k(\alpha)$, and the quotient $y_k(\alpha) := n/q_k(\alpha)$, and

$$S(\alpha, n) := \frac{R_\alpha(n) + 1}{n} = 1 + \frac{1 + x_k(\alpha)}{y_k(\alpha)}. \tag{1}$$

As $y_k(\alpha)$ belongs to the interval $[x_k(\alpha), 1]$, the following bounds hold

$$2 + x_k(\alpha) \leq \frac{R_\alpha(n) + 1}{n} \leq 2 + \frac{1}{x_k(\alpha)} \tag{2}$$

(the lower bound holds for n close to $q_k(\alpha)$ whereas the upper bound is attained for $n = q_{k-1}(\alpha)$).

The ratio $x_k(\alpha)$ belongs to $]0, 1]$, and the Borel-Bernstein Theorem (see e.g. [6]) proves that $\liminf_{k \rightarrow \infty} x_k(\alpha) = 0$ for almost any irrational α . More precisely:

Proposition 3. (i) *For any irrational real α , one has*

$$\liminf_{n \rightarrow \infty} \frac{R_\alpha(n)}{n} \leq 3.$$

(ii) [Morse and Hedlund] [9] *For almost any irrational α , one has*

$$\limsup_{n \rightarrow \infty} \frac{R_\alpha(n)}{n \log n} = +\infty, \text{ and } \limsup_{n \rightarrow \infty} \frac{R_\alpha(n)}{n(\log n)^{1+\varepsilon}} = 0 \text{ for } \varepsilon > 0.$$

3 Probabilistic Model and Main Results

The two extreme bounds in Eq. (2) may be very different, notably when $x_k(\alpha)$ is small. We wish to study the behaviour of the ratio $S(\alpha, n)$ when n is any integer in $\llbracket q_{k-1}(\alpha), q_k(\alpha) - 1 \rrbracket$. Equation (1) shows the role of the quotient n/q_k (called $y_k(\alpha)$ there) and leads to the notion of *position*.

3.1 Position

We consider a fixed sequence $(\mu_k)_k$ with values in $[0, 1[$, and for each $\alpha \in \mathcal{I} :=]0, 1]$, and each $k \in \mathbb{N}$, we consider the real number at (barycentric) position μ_k inside the interval $\llbracket q_{k-1}(\alpha), q_k(\alpha) - 1 \rrbracket$, namely

$$\tilde{n}_k^{\langle \mu_k \rangle}(\alpha) := q_{k-1}(\alpha) + \mu_k(q_k(\alpha) - q_{k-1}(\alpha)),$$

together with its integer part (which belongs to $\llbracket q_{k-1}(\alpha), q_k(\alpha) - 1 \rrbracket$),

$$n_k^{\langle \mu_k \rangle}(\alpha) = \lfloor \tilde{n}_k^{\langle \mu_k \rangle}(\alpha) \rfloor = q_{k-1}(\alpha) + \lfloor \mu_k(q_k(\alpha) - q_{k-1}(\alpha)) \rfloor.$$

The subsequence $(n_k^{\langle \mu_k \rangle}(\alpha))_k$ is *the subsequence associated with the positions μ_k* .

We are interested in the subsequence of $n \mapsto S(\alpha, n)$ associated with the subsequence $\{n_k^{\langle \mu_k \rangle}(\alpha), k \in \mathbb{N}\}$, and we then let $S_k^{\langle \mu_k \rangle}(\alpha) := S(\alpha, n_k^{\langle \mu_k \rangle}(\alpha))$, namely

$$S_k^{\langle \mu_k \rangle}(\alpha) = 1 + \frac{q_{k-1}(\alpha) + q_k(\alpha)}{n_k^{\langle \mu_k \rangle}(\alpha)} = 1 + \frac{q_{k-1}(\alpha) + q_k(\alpha)}{q_{k-1}(\alpha) + \lfloor \mu_k(q_k(\alpha) - q_{k-1}(\alpha)) \rfloor}. \tag{3}$$

If we drop the integer part in the expression of $S_k^{(\mu_k)}$, we deal with the sequence $\tilde{S}_k^{(\mu_k)}(\alpha) := S(\alpha, \tilde{n}_k^{(\mu_k)}(\alpha))$, namely,

$$\tilde{S}_k^{(\mu_k)}(\alpha) = 1 + \frac{q_{k-1}(\alpha) + q_k(\alpha)}{\tilde{n}_k^{(\mu_k)}(\alpha)} = 1 + \frac{q_{k-1}(\alpha) + q_k(\alpha)}{q_{k-1}(\alpha) + \mu_k(q_k(\alpha) - q_{k-1}(\alpha))}, \quad (4)$$

which is expressed with the two sequences $(x_k(\alpha))_k$ and (μ_k) as

$$\tilde{S}_k^{(\mu_k)}(\alpha) = f_{\mu_k}(x_k(\alpha)) \quad \text{with} \quad f_\mu(x) := 1 + \frac{1+x}{x + \mu(1-x)}. \quad (5)$$

The study of the function f_μ provides a precise knowledge on the sequence $\tilde{S}_k^{(\mu_k)}(\alpha)$, that may be “tranfered” to the sequence $S_k^{(\mu_k)}(\alpha)$ since the two sequences are “close enough”. The following result provides such a first instance of this strategy:

Proposition 4. *Consider a sequence $(\mu_k)_k$ with $\mu_k \in [0, 1]$, and let $\alpha \in [0, 1] \setminus \mathbb{Q}$.*

(i) *Denote by m_k the k -th partial quotient of α . Then, $x_k(\alpha) \leq 1/(m_k + 1)$ and*

$$\tilde{S}_k^{(\mu_k)}(\alpha) \in \left[1 + \frac{m_k + 2}{\mu_k m_k + 1}, 3 \right] \quad \text{or} \quad \tilde{S}_k^{(\mu_k)}(\alpha) \in \left[3, 1 + \frac{m_k + 2}{\mu_k m_k + 1} \right]$$

depending whether $\mu_k \in [1/2, 1]$ or $\mu_k \in [0, 1/2]$.

(ii) *The sequence $S_k^{(\mu_k)}(\alpha)$ is bounded if α has bounded partial quotients or if the sequence (μ_k) admits a strictly positive lower bound.*

Proof. The map $f_\mu : [0, 1] \rightarrow \mathbb{R}$ is strictly decreasing when $\mu \in]0, 1/2[$, and strictly increasing when $\mu \in]1/2, 1[$. This is the constant function equal to 3 when $\mu = 1/2$. For any $a \in]0, 1[$, the image $f_\mu([a, 1])$ is the interval with endpoints 3 and $f_\mu(a)$. This proves Assertion (i).

With the two inequalities

$$\tilde{n}_k^{(\mu)} \geq n_k^{(\mu)} \geq q_{k-1} \geq \varphi^{1-k}, \quad 0 \leq \tilde{n}_k^{(\mu)} - n_k^{(\mu)} \leq 1,$$

we obtain the inequality

$$0 \leq S_k^{(\mu)} - \tilde{S}_k^{(\mu)} = \frac{q_k + q_{k-1}}{n_k^{(\mu)} \cdot \tilde{n}_k^{(\mu)}} (\tilde{n}_k^{(\mu)} - n_k^{(\mu)}) \leq \frac{1}{q_{k-1}} \frac{q_k + q_{k-1}}{\tilde{n}_k^{(\mu)}} \leq \varphi^{k-1} \tilde{S}_k^{(\mu)}, \quad (6)$$

and we apply (i).

3.2 Probabilistic Model

Let us describe now our *probabilistic model*. We choose a sequence $(\mu_k)_k$ of positions that will be *fixed*. This defines, for each real α , a sequence of indices $n_k := n_k^{(\mu_k)}$, and then a sequence of real numbers $k \mapsto S_k^{(\mu_k)}(\alpha)$. When the real

α is random, and uniformly drawn in the unit interval $\mathcal{I} = [0, 1]$, the sequence $k \mapsto S_k^{(\mu_k)}$ becomes a sequence of random variables, and we study the mean value and the distribution of the sequence $k \mapsto S_k^{(\mu_k)}$ for $k \rightarrow \infty$.

For any position, the index $n_k^{(\mu_k)}$ belongs to the interval $[[q_{k-1}, q_k - 1]]$. Then, as the expectations for $\alpha \in [0, 1]$ of the two extreme sequences $k \mapsto \log q_{k-1}(\alpha)$, $k \mapsto \log q_k(\alpha)$ satisfy the same estimates (see [7]), it is also the case for the expectation for $\alpha \in [0, 1]$ of the sequence $k \mapsto \log n_k^{(\mu_k)}(\alpha)$. It thus satisfies

$$\mathbb{E}[\log n_k^{(\mu_k)}] = \frac{\pi^2}{12 \log 2} k + O(1), \tag{7}$$

and it is of linear growth with respect to k .

3.3 Results for a Constant Position μ

We first consider the case where the sequence $(\mu_k)_k$ is a constant sequence that takes a fixed value μ , and we study the expectation and the distribution of the sequence $k \mapsto S_k^{(\mu)}$ of random variables, when $k \rightarrow \infty$, as a function of the position μ . Theorem 1 below shows that there are two main cases (Fig. 2):

- (a) the case when $\mu = 0$; here, the expectations are infinite, but the functions $k \mapsto S_k^{(0)}$ admit a limit density;
- (b) the case when $\mu \neq 0$; here, both the expectations and the densities have a finite limit; the case $\mu = 1/2$ is particular, as the limit density is a Dirac measure, concentrated at the value 3.

For indices n associated with parameters μ satisfying $\mu \geq \mu_0 > 0$, we exhibit a behaviour for the sequence $n \mapsto R_\alpha(n)$ which is thus “linear on average”; the “log n ” behaviour of Proposition 3 does not occur in this case.

Theorem 1 (Fixed position μ). *Let $\varphi = (\sqrt{5} - 1)/2 < 1$. The following holds for the random variables $S_k^{(\mu)}$.*

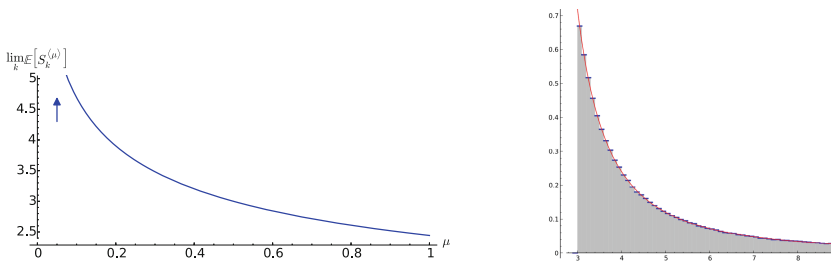


Fig. 2. On the left, the graph of $\lim_{k \rightarrow \infty} \mathbb{E}[S_k^{(\mu)}]$ as a function of μ . On the right, the graph of the density $s^{(0)}$.

(i) [Expectations] For each $\mu \in]0, 1]$, their expected values $\mathbb{E}[S_k^{(\mu)}]$ satisfy

$$\mathbb{E}[S_k^{(\mu)}] = 1 + \frac{1}{\log 2} \frac{|\log \mu|}{1 - \mu} + O\left(\frac{\varphi^{2k}}{\mu}\right) + O\left(\varphi^k \frac{|\log \mu|}{1 - \mu}\right), \quad (8)$$

with the constants in the O -term being uniform with respect to μ and k .

(ii) [Limit density] For each $\mu \in [0, 1]$ with $\mu \neq 1/2$, they admit a limit density $s^{(\mu)}$ equal to

$$s^{(\mu)}(x) = \frac{1}{\log 2} \left(\frac{1}{(x-1)|x(1-\mu) + \mu - 2|} \right) \mathbf{1}_{\mathcal{I}_\mu}(x), \quad (9)$$

where \mathcal{I}_μ is the real interval with endpoints 3 and $1 + 1/\mu$.

More precisely, for any $b \in I_\mu$, one has

$$\mathbb{P}\left[S_k^{(\mu)} \leq b\right] = \int_0^b s^{(\mu)}(x) dx + \frac{1}{b} O(\varphi^k),$$

where the constant of the O -term is uniform with respect to b and k . It is also uniform with respect to μ when μ satisfies $|\mu - 1/2| \geq \mu_0$ for any $\mu_0 > 0$.

3.4 Results When the Sequence $\mu_k \rightarrow 0$

We now focus on the difficult case, when the sequence $(\mu_k)_k$ is no longer constant, and we consider a sequence $(\mu_k)_k$ of positions which tends to 0. We first consider in Theorem 2 below sequences $(\mu_k)_k$ which tend exponentially fast to 0, and we observe that the expectations are of order k . We then consider general sequences $(\mu_k)_k$ which tend to 0, and we show that the associated random variables admit a limit density, with a speed of convergence which depends on the sequence $(\mu_k)_k$.

Theorem 2 (Sequence $\mu_k \rightarrow 0$). *The following holds for the random variables $S_k^{(\mu)}$ associated with a sequence $\mu_k \rightarrow 0$.*

(i) [Expectations] Consider the sequence $\mu_k = \tau^k$, with $\tau \in [\varphi^2, 1[$. Then

$$\mathbb{E}[S_k^{(\tau^k)}] = k \frac{|\log \tau|}{\log 2} + O(1), \quad (10)$$

where the constant hidden in the O -term is uniform with respect to τ and k . For any α , and for each $\tau \in [\varphi^2, 1[$, there exists an increasing subsequence $\mathcal{N}(\alpha, \tau)$ of indices n for which

$$\mathbb{E}\left[\frac{R_\alpha(n)}{n} - \frac{12|\log \tau|}{\pi^2} \log n\right] = O(1) \quad (n \rightarrow \infty). \quad (11)$$

For any $\tau < 1$, if μ_k is drawn uniformly in $[0, 1]$, the conditional expectation with respect to the event $[\mu_k \geq \tau^k]$ satisfies

$$\lim_{k \rightarrow \infty} \mathbb{E}\left[S_k^{(\mu_k)} \mid [\mu_k \geq \tau^k]\right] = 1 + \frac{\pi^2}{6 \log 2}.$$

(ii) [Limit density] For any sequence $\mu_k \rightarrow 0$, the random variables $S_k^{(\mu_k)}$ admit as limit density the density $s^{(0)}$ equal to

$$s^{(0)}(x) = \frac{1}{\log 2} \frac{1}{(x-1)(x-2)} \mathbf{1}_{[3,\infty)}(x).$$

More precisely, for any $b \geq 3$, the probability $\mathbb{P}[S_k^{(\mu_k)} \geq b]$ satisfies

$$\mathbb{P}[S_k^{(\mu_k)} \geq b] = \frac{1}{\log 2} \log \left(\frac{b-1}{b-2} \right) + O(\mu_k) + \frac{1}{b} O(\varphi^k),$$

where the constants hidden in the O -term are uniform to respect to b and k . If now the sequences $(b_k)_k$ and $(\mu_k)_k$ satisfy the following three conditions ($b_k \rightarrow \infty$, $\mu_k \rightarrow 0$ with $b_k \mu_k \rightarrow 0$), then

$$\lim_{k \rightarrow \infty} b_k \cdot \mathbb{P} \left[S_k^{(\mu_k)} \geq b_k \right] = \frac{1}{\log 2}.$$

Remark 1. The estimate (7) together with (10) yields (11). We have then exhibited a $\log n$ behaviour “on average” for the ratio $R_\alpha(n)/n$ for (an infinity of) particular subsequences n (which depend on α). On the contrary, when the position is not too small, the ratio $R_\alpha(n)/n$ remains bounded (on average).

4 Strategy for the Proofs.

We begin with Theorem 1 which deals with a fixed position μ . There are three main steps in the proof of Theorem 1.

- (i) We drop the integer part in the expression of $S_k^{(\mu)}$ and deal with the sequence $\tilde{S}_k^{(\mu)}(\alpha)$ that can be written as $f_\mu(x_k(\alpha))$ (see (5)). This is an instance of a smooth sequence (as defined in Sect. 4.1 below). We express its mean value and its distribution with the k -th iterate of the Perron Frobenius operator \mathbf{H} .
- (ii) With the spectral properties of the operator \mathbf{H} (described in Sect. 4.2), when acting on the Banach space $BV(\mathcal{I})$ of the functions of bounded variation on the unit interval \mathcal{I} , we obtain the asymptotics of the expectations and the expression of the limit distribution, always for the sequence $\tilde{S}_k^{(\mu)}$.
- (iii) We return to the initial sequence $S_k^{(\mu)}$ with the following estimates

$$\mathbb{E}[S_k^{(\mu)}] = \mathbb{E}[\tilde{S}_k^{(\mu)}] (1 + O(\varphi^k)), \quad \mathbb{P}[S_k^{(\mu)} \leq b] - \mathbb{P}[\tilde{S}_k^{(\mu)} \leq b] = O\left(\frac{\varphi^k}{b}\right), \tag{12}$$

which are refinements of Eq. (6) and will be proven in an extended version.

Since the probabilistic estimates obtained in Theorem 1 are uniform with respect to μ and k , we may extend them to the case where μ depends on k , and we may study the interesting case where the sequence $(\mu_k)_k$ tends to 0 for $k \rightarrow \infty$. We then obtain the results of Theorem 2.

Remark 2. There are two error terms in the asymptotic estimates (8) of the expectations. The first one comes from the spectral gap of the Perron-Frobenius operator and the second one arises when one takes into account integer parts in the definition of $S_k^{(\mu)}$.

4.1 Smooth Sequences

The sequence $\widetilde{S}_k^{(\mu)}$ provides an instance of a smooth sequence, defined as follows:

Definition 1. A sequence of random variables (T_k) defined on the unit interval $\mathcal{I} = [0, 1]$ is a *smooth sequence* if there exists a function $f \in BV(\mathcal{I})$ for which

$$T_k(\alpha) = f(x_k(\alpha)) \quad \text{with} \quad x_k(\alpha) = \frac{q_{k-1}(\alpha)}{q_k(\alpha)} \quad \text{for all } \alpha \in \mathcal{I}.$$

Here, we deal with the function f_μ defined in (5), whose inverse map g_μ is

$$g_\mu : f_\mu(\mathcal{I}) \mapsto [0, 1], \quad g_\mu(x) = \frac{-1 - \mu + \mu x}{2 - \mu - x(1 - \mu)}.$$

For $\mu \in]0, 1[$, the function f_μ is integrable on \mathcal{I} , and its L^1 -norm satisfies

$$\|f_\mu\|_{L^1} = 1 + \frac{1}{1 - \mu} + \frac{1 - 2\mu}{(1 - \mu)^2} |\log \mu|, \quad \|f_1\|_{L^1} = 5/2.$$

Moreover, always for $\mu \in]0, 1[$, the function f_μ is monotonic and thus of bounded variation, with a total variation equal to $(1/\mu)|1 - 2\mu|$, hence $\|f_\mu\|_{BV} = O(1/\mu)$. Remark that f_0 does not belong to $BV(\mathcal{I})$.

We now recall some basic facts on the underlying dynamical system, together with the Perron-Frobenius operator, that will be useful in the sequel.

4.2 The Dynamical System and the Perron-Frobenius Operator

The Underlying Dynamical System. We consider the dynamical system (\mathcal{I}, V) associated with the unit interval \mathcal{I} and the Gauss map V , defined by

$$V(x) = \frac{1}{x} - \left\lfloor \frac{1}{x} \right\rfloor = \left\{ \frac{1}{x} \right\} \quad \text{for } x \neq 0, \quad V(0) = 0.$$

The map V builds the continued fraction expansion of α , via the function $m(\alpha) := \lfloor 1/\alpha \rfloor$, as

$$\alpha = [m_1, m_2, \dots, m_k, \dots] \quad \text{with} \quad m_{k+1}(\alpha) = m(V^k(\alpha)) \quad \text{for all } k \geq 0.$$

The inverse branches of V belong to the set

$$\mathcal{H} := \left\{ h_m : x \mapsto \frac{1}{m + x}; \quad m \geq 1 \right\},$$

and the inverse branches of V^k belong to the set

$$\mathcal{H}^k = \{ h_{m_1} \circ h_{m_2} \circ \dots \circ h_{m_k} \quad : \quad m_1, \dots, m_k \geq 1 \}.$$

For a k -uple $\mathbf{m} = (m_1, m_2, \dots, m_k)$, let $h_{\mathbf{m}} := h_{m_1} \circ h_{m_2} \circ \dots \circ h_{m_k}$. The linear fractional transformation $h_{\mathbf{m}}$ is expressed with two sequences of continuants $(p_k)_k, (q_k)_k$ under the form

$$h_{\mathbf{m}}(x) = h_{m_1} \circ h_{m_2} \circ \dots \circ h_{m_k}(x) = \frac{1}{m_1 + \frac{1}{\ddots + \frac{1}{m_k + x}}} = \frac{p_{k-1}x + p_k}{q_{k-1}x + q_k}.$$

Remark that the continuants q_k, p_k which are just defined only depend on the k -uple $\mathbf{m} = (m_1, m_2, \dots, m_k)$. However, there is no conflict with our previous definition of the sequence $q_k(\alpha)$ given in Sect. 2, since, for any α which belongs to the interval $h_{\mathbf{m}}(\mathcal{I})$, the equality $q_k(\alpha) = q_k(\mathbf{m})$ holds.

The mirror property (described for instance in [1]) relates the coefficients of $h = h_{m_1} \circ h_{m_2} \circ \dots \circ h_{m_k}$ and those of its mirror $\widehat{h} := h_{m_k} \circ h_{m_{k-1}} \circ \dots \circ h_{m_1}$:

$$h(y) = \frac{p_{k-1}y + p_k}{q_{k-1}y + q_k} \implies \widehat{h}(y) = \frac{p_{k-1}y + q_{k-1}}{p_k y + q_k}.$$

Perron-Frobenius Operator. When the unit interval is endowed with a density f , after one iteration of V , it is endowed with the density

$$\mathbf{H}[f](x) := \sum_{h \in \mathcal{H}} |h'(x)| \cdot f \circ h(x),$$

and after k iterations of V , with the density

$$\mathbf{H}^k[f](x) = \sum_{h \in \mathcal{H}^k} |h'(x)| \cdot f \circ h(x).$$

The operator \mathbf{H} is called the Perron Frobenius operator.

Now, at $x = 0$, the two maps h and \widehat{h} satisfy $|h'(0)| = |\widehat{h}'(0)| = 1/q_k^2$, and the equality $q_{k-1}/q_k = \widehat{h}(0)$ holds. With this remark, the k -th iterate \mathbf{H}^k generates the continuants q_k ,

$$\mathbf{H}^k[f](0) = \sum_{h \in \mathcal{H}^k} \frac{1}{q_k^2} f\left(\frac{p_k}{q_k}\right) = \sum_{h \in \mathcal{H}^k} \frac{1}{q_k^2} f\left(\frac{q_{k-1}}{q_k}\right). \tag{13}$$

We now summarize some classical spectral properties of the operator \mathbf{H} (see e.g. [6] or [2]). When acting on the Banach space $BV(\mathcal{I})$ of functions of bounded variation, the operator \mathbf{H} admits a unique dominant eigenvalue $\lambda = 1$, with an eigenfunction proportional to $\psi(x) = 1/(1+x)$, and it has a subdominant spectral radius equal to φ^2 . Moreover, the adjoint \mathbf{H}^* has an eigenmeasure proportional to the Lebesgue measure. Then, for any $g \in BV(\mathcal{I})$, the iterate $\mathbf{H}^k[g]$ decomposes as

$$\mathbf{H}^k[g](x) =? \frac{1}{\log 2} \frac{1}{1+x} \cdot \int_{\mathcal{I}} g(x) dx + O(\varphi^{2k}) \|g\|_{BV}. \tag{14}$$

4.3 Smooth Random Variables and Perron-Frobenius Operator

We now perform Step (i) in the proof of Theorem 1. The following lemma (inspired by [5]) expresses the expectation and distribution of smooth sequences in terms of the Perron-Frobenius operator \mathbf{H} .

Lemma 1. *Assume that (T_k) is a smooth sequence associated with the function f . Then, the expected value $\mathbb{E}[T_k]$ and the distribution of the random variable (T_k) are both expressed with the k -th iterate of the Perron-Frobenius operator \mathbf{H} :*

$$\mathbb{E}[T_k] = \mathbf{H}^k \left[f(x) \cdot \frac{1}{1+x} \right] (0), \quad \mathbb{P}[T_k \in J] = \mathbf{H}^k \left[\mathbf{1}_J \circ f(x) \cdot \frac{1}{1+x} \right] (0),$$

where J is a subinterval of \mathcal{I} .

Proof. For each index k , consider the family of linear fractional transformations $h \in \mathcal{H}^k$. The intervals $h(\mathcal{I})$ form a partition of the interval \mathcal{I} , and the length of the interval $h(\mathcal{I})$ is expressed as a function of the continuants q_k , as

$$|h(\mathcal{I})| = \frac{1}{q_k(q_k + q_{k-1})} = \frac{1}{q_k^2} \left(\frac{1}{1 + \frac{q_{k-1}}{q_k}} \right).$$

Moreover $T_k(\alpha)$ is constant on the interval $h(\mathcal{I})$, and equal to $f(q_{k-1}/q_k)$. Finally

$$\mathbb{E}[T_k] := \int_{\mathcal{I}} T_k(\alpha) d\alpha = \sum_{h \in \mathcal{H}^k} \frac{1}{q_k^2} \ell \left(\frac{q_{k-1}}{q_k} \right) \quad \text{with} \quad \ell(x) = \frac{1}{1+x} f(x).$$

With Relation (13), the last expression is exactly $\mathbf{H}^k[\ell](0)$.

We now consider, for any $J \subset \mathbb{R}$, the probability $\mathbb{P}[T_k \in J] = \mathbb{E}[\mathbf{1}_J \circ T_k]$. Using the same transforms as above (now applied to the function $\mathbf{1}_J \circ f(x)$) yields

$$\mathbb{P}[T_k \in J] = \mathbf{H}^k \left[\mathbf{1}_J \circ f(x) \cdot \frac{1}{1+x} \right] (0).$$

4.4 Asymptotic Study of Smooth Variables

We now perform Step (ii) in the proof of Theorem 1. Since the probabilistic characteristics of the random variable T_k are expressed with the k -th iterate of the Perron Frobenius operator \mathbf{H} , their asymptotics will be related to the dominant spectral properties of this operator when it acts on the Banach space $BV(\mathcal{I})$ of the functions of bounded variation on the unit interval, and we use the decomposition (14).

Lemma 2. *The following asymptotics hold, for any smooth sequence (T_k) relative to a function $f \in BV(\mathcal{I})$:*

$$\mathbb{E}[T_k] = \frac{1}{\log 2} \int_{\mathcal{I}} f(x) \cdot \frac{1}{1+x} dx + O(\varphi^{2k} \|f\|_{BV}),$$

$$\mathbb{P}[T_k \in J] = \frac{1}{\log 2} \int_{\mathcal{I}} \mathbf{1}_J \circ f(x) \cdot \frac{1}{1+x} dx + O(\varphi^{2k}),$$

where J is a subinterval of \mathcal{I} . If moreover the function f is of class \mathcal{C}^1 and monotonic, with an inverse function g , the random variable T_k admits a limit density; for any interval $[a, b] \subset f(\mathcal{I})$, one has

$$\mathbb{P}[T_k \in [a, b]] = \frac{1}{\log 2} \int_a^b \frac{|g'(u)|}{1+g(u)} du + O(\varphi^{2k}) = \frac{1}{\log 2} \left| \log \frac{1+g(a)}{1+g(b)} \right| + O(\varphi^{2k}).$$

Proof. This is just an easy application of the decomposition (14). For the distribution, the norm $\|\mathbf{1}_J \circ f \cdot \psi\|_{BV}$ admits an upper bound which neither depend on the function f nor on the interval J .

The previous lemma entails the following asymptotics for the probabilistic characteristics of the sequence $\tilde{S}_k^{(\mu)}$. Recall that the density $s^{(\mu)}$ is defined in (9).

Lemma 3. For $\mu \in]0, 1]$, the two following asymptotic estimates, namely

$$\mathbb{E}[\tilde{S}_k^{(\mu)}] = 1 + \frac{1}{\log 2} \frac{|\log \mu|}{1-\mu} + O\left(\frac{\varphi^{2k}}{\mu}\right), \quad \mathbb{P}[\tilde{S}_k^{(\mu)} \in J] = \int_J s^{(\mu)}(x) dx + O(\varphi^{2k}).$$

The second estimate also holds for $\mu = 0$.

Proof. This is just the application of the previous lemma for $f := f_\mu$. The function f_μ belongs to $BV(\mathcal{I})$ for $\mu > 0$, with norm $\|f_\mu\|_{BV} = O(1/\mu)$.

This ends Step (ii) of the proof of Theorem 1. The estimates (12) needed in Step (iii) will be proven in the extended version, together with some hints for Theorem 2.

5 Conclusion

With a Sturmian word of slope α , and a sequence (μ_k) , we have associated a sequence of indices n_k (lengths of factors) defined by their barycentric position μ_k inside $[[q_{k-1}(\alpha), q_k(\alpha) - 1]]$. We then have elucidated the role played by the position in the behaviour of the recurrence of a random Sturmian word.

We plan to extend our probabilistic study in three directions, and consider three probabilistic models. The first two models were already dealt with in [4] for the study of Kronecker sequences, and the last one has been considered in [10].

Reals with Bounded Partial Quotients. This type of slope α gives rise to Sturmian words whose recurrence function is proven to be linear (see Lemma 4). For a bound M , we restrict α to the set $\mathcal{R}^{[M]}$ of numbers whose partial quotients are at most M , endowed with the Hausdorff measure, and we wish to observe the transition when $M \rightarrow \infty$.

Rational Numbers. This type of slope α gives rise to periodic words, and occurs for Christoffel words. For a bound N , we restrict α to the set $\mathcal{Q}_{[N]}$ of rationals with denominator at most N , endowed with the uniform distribution, and we wish to observe the transition when $N \rightarrow \infty$. This will explain how a periodic word “becomes” Sturmian.

Quadratic Irrationals. This type of slope α occurs for substitutive Sturmian words. There is a natural notion of size associated with such numbers α , closely related to the period of their continued fraction expansion, and we wish to observe the transition when the size tends to ∞ .

References

1. Adamczewski, B., Allouche, J.-P.: Reversals and palindromes in continued fractions, *heuret. Comput. Sci.* **380**, 220–237 (2007)
2. Bourdon, J., Daireaux, B., Vallée, B.: Dynamical analysis of α -Euclidean algorithms. *J. Algorithms* **44**, 246–285 (2002)
3. Cassaigne, J.: Limit values of the recurrent quotient of Sturmian sequences. *Theoret. Comput. Sci.* **218**, 3–12 (1999)
4. Cesaratto, E., Vallée, B.: Pseudo-randomness of a random Kronecker sequence. An instance of dynamical analysis, Chapter 11. In: Berthé, V., Rigo, M. (eds.) *Combinatorics, Words and Symbolic Dynamics* in the book *Combinatorics, Words and Symbolic Dynamics*, pp. 405–448. Cambridge University Press (To appear)
5. Flajolet, P., Vallée, B.: Continued fraction algorithms, functional operators, and structure constants. *Theoret. Comput. Sci.* **94**, 1–34 (1998)
6. Losifescu, M., Kraaikamp, C.: *Metrical Theory of Continued Fractions*, Collection Mathematics and Its Applications. Kluwer Academic Press, Dordrecht (2002)
7. Lévy, P.: Sur le développement en fraction continue d’un nombre choisi au hasard. *Compos. Math.* **3**, 286–303 (1936)
8. Lothaire, M.: *Algebraic Combinatorics on Words*. Encyclopedia of Mathematics and Its Applications. Cambridge University Press, Cambridge (2002)
9. Morse, M., Hedlund, G.: Symbolic dynamics II. Sturmian trajectories. *Am. J. Math.* **62**, 1–42 (1940)
10. Vallée, B.: Dynamique des fractions continues à contraintes périodiques. *J. Number Theor.* **72**(2), 183–235 (1998)

Exponential-Size Model Property for PDL with Separating Parallel Composition

Joseph Boudou^(✉)

IRIT, Toulouse University, Toulouse, France
joseph.boudou@irit.fr

Abstract. Propositional dynamic logic is extended with a parallel program having a separating semantic: the program $(\alpha \parallel \beta)$ executes α and β on two substates of the current state. We prove that when the composition of two substates is deterministic, the logic has the exponential-size model property. The proof is by a piecewise filtration using an adaptation of the Fischer-Ladner closure. We conclude that the satisfiability of the logic is decidable in NEXPTIME.

1 Introduction

Propositional dynamic logic (PDL) is a multi-modal logic designed to reason about behaviors of programs [10]. With each program α we associate a modal operator $[\alpha]$, formulas $[\alpha]\varphi$ being read “all executions of α from the current state lead to a state where φ holds”. The set of programs is structured by some operators: sequential composition $(\alpha ; \beta)$ of programs α and β executes β after α ; nondeterministic choice $(\alpha \cup \beta)$ of program α and β executes α or β , nondeterministically; test $\varphi?$ on formula φ does nothing but can be executed only if the current state satisfies φ ; iteration α^* of program α executes α a nondeterministic number of times. A limitation of PDL is the lack of a construct to reason about concurrency.

Different extensions of PDL have been devised to overcome this limitation, for instance interleaving PDL [1], PDL with intersection [11] and the concurrent dynamic logic [16]. PDL with storing, recovering and parallel composition (PRSPDL) [4] is another extension of PDL for concurrency. The key difference is that in PRSPDL, the program $(\alpha \parallel \beta)$ executes α and β in parallel *on two substates* of the current state. Hence, $(\alpha \parallel \beta)$ being executable at some state does not imply that α or β is executable at that state. Moreover, since states can be separated in substates (and merged back), PRSPDL is related to the Boolean logic of bunched implication (BBI) [17]. Indeed, a multiplicative conjunction semantically similar to the one found in BBI can be defined in PRSPDL. Thus PRSPDL can be compared to the classical version of the multi-modal logic of bunched implication (MBIc) [7], a difference being that MBIc has only the parallel composition as program constructs, limiting its expressive power [2].

This work was supported by the “French National Research Agency” (DynRes contract ANR-11-BS02-011).

The combination of separation and concurrency provided by the parallel construct of PRSPDL suggests some interesting applications. For instance, a dynamic and concurrent logic on heaps of memory akin to separation logics [8,18], may be envisioned. Moreover, as PDL has been adapted to different contexts, a separating parallel composition may be of interest in some of them. For instance, in dynamic epistemic logics [9], the parallel epistemic action $(! \varphi \parallel ! \psi)$ could mean that φ is announced to a group of agents and ψ is announced to the agents not in that group. Despite this potential, there has been almost no complexity analysis of dynamic logic with separating parallel composition. The complexity of PRSPDL is not studied in [4]. In [3], PRSPDL interpreted over frames where there is at most one decomposition of any state into substates is proved to be highly undecidable.

In this paper, we study the complexity of PDL with separating parallel composition (PPDL). The language of this logic is the fragment of PRSPDL without the store and recover programs which allow to access substates directly. We focus on the class of \triangleleft -deterministic frames where the composition of substates is deterministic: there is at most one way to merge two states. This restriction is quite natural and has been studied in many logics with separation like separation logics, ambient logics [6], BBI [13] and arrow logics [14]. We show that for PPDL, \triangleleft -determinism conveys some interesting properties, notably a strong finite model property leading to a complexity upper bound of NEXPTIME for the satisfiability problem. This result contrasts with the 2EXPTIME complexity of other dynamic logics with concurrency like PDL with intersection [12] or interleaving PDL [15]. To prove this result, we provide nontrivial adaptations of existing methods (Fischer-Ladner closure and model unraveling) along with some new concepts (placeholders, marking functions and the neat model property).

The paper is structured as follows. In the next section, the language and semantic of PPDL is formally defined. In Sect. 3, the problem of decomposing formulas of the forms $[\alpha \parallel \beta] \varphi$ is resolved by extending the language and by adapting the Fischer-Ladner closure. In Sect. 4, the new concepts of threads, twines and neat models are introduced. And in Sect. 5, it is proved that the class of neat \triangleleft -deterministic frames satisfies the same formulas as the class of \triangleleft -deterministic frames. In Sect. 6, the strong finite model property is proved by piecewise filtration.

2 Propositional Dynamic Logic with Separating Parallel Composition (PPDL)

Let Π_0 be a countable set of atomic programs (denoted by a, b, \dots) and Φ_0 a countable set of propositional variables (denoted by p, q, \dots). The sets Π and Φ of programs and formulas are defined by:

$$\begin{aligned} \alpha, \beta &:= a \mid (\alpha ; \beta) \mid (\alpha \cup \beta) \mid \varphi? \mid \alpha^* \mid (\alpha \parallel \beta) \\ \varphi &:= p \mid \perp \mid \neg \varphi \mid [\alpha] \varphi \end{aligned}$$

The negation construct is an involution: by definition, $\neg\neg\varphi = \varphi$. Parentheses may be omitted for clarity, but they are taken into account when counting occurrences of symbols. We write $|\alpha|$ and $|\varphi|$ for the number of occurrences of symbols in the program α and the formula φ , respectively. We define the abbreviations $\top \doteq \neg\perp$ and $\langle\alpha\rangle\varphi \doteq \neg[\alpha]\neg\varphi$. The missing usual (additive) Boolean operators can be defined too, starting with $\varphi \rightarrow \psi \doteq [\varphi?]\psi$. Additionally, a multiplicative conjunction related to BBI [17] may be defined as $\varphi * \psi \doteq \langle\varphi? \parallel \psi?\rangle\top$.

A frame is a tuple (W, R, \triangleleft) where W is a non-empty set of states (denoted by w, x, y, \dots), R is a function associating a binary relation over W to each atomic program and \triangleleft is a ternary relation over W . Intuitively, $x R(a) y$ means that the program a can be executed in state x , reaching state y . Similarly, $x \triangleleft (y, z)$ means that x can be split into the substates y and z or equivalently that y and z can be merged to obtain x . When the merging of states is functional, the frame is said to be \triangleleft -deterministic. This is a common restriction, for instance in separation logics, expressing the fact that the parts determine the whole. Formally, a frame is \triangleleft -deterministic iff for all $x, y, w_1, w_2 \in W$, if $x \triangleleft (w_1, w_2)$ and $y \triangleleft (w_1, w_2)$ then $x = y$. The class of \triangleleft -deterministic frames is denoted by $\mathcal{C}_{\triangleleft\text{-det}}$.

A model is a tuple (W, R, \triangleleft, V) where (W, R, \triangleleft) is a frame and V is a function associating a subset of W to each propositional variable. A model is \triangleleft -deterministic iff its frame is \triangleleft -deterministic. The forcing relation \models is defined by parallel induction along with the extension of R to all programs:

$$\begin{aligned}
 \mathcal{M}, x \models p & \quad \text{iff } x \in V(p) \\
 \mathcal{M}, x \models \perp & \quad \text{never} \\
 \mathcal{M}, x \models \neg\varphi & \quad \text{iff } \mathcal{M}, x \not\models \varphi \\
 \mathcal{M}, x \models [\alpha]\varphi & \quad \text{iff } \forall y \in W, \text{ if } x R(\alpha) y \text{ then } \mathcal{M}, y \models \varphi \\
 x R(\alpha; \beta) y & \quad \text{iff } \exists z \in W, x R(\alpha) z \text{ and } z R(\beta) y \\
 x R(\alpha \cup \beta) y & \quad \text{iff } x R(\alpha) y \text{ or } x R(\beta) y \\
 x R(\varphi?) y & \quad \text{iff } x = y \text{ and } \mathcal{M}, x \models \varphi \\
 x R(\alpha^*) y & \quad \text{iff } x R(\alpha)^* y \\
 & \quad \text{where } R(\alpha)^* \text{ is the reflexive and transitive closure of } R(\alpha) \\
 x R(\alpha \parallel \beta) y & \quad \text{iff } \exists w_1, w_2, w_3, w_4 \in W, \\
 & \quad x \triangleleft (w_1, w_2), w_1 R(\alpha) w_3, w_2 R(\beta) w_4 \text{ and } y \triangleleft (w_3, w_4)
 \end{aligned}$$

Given a class \mathcal{C} of frames, a formula φ is *satisfiable in \mathcal{C}* iff there exists a model $\mathcal{M} = (W, R, \triangleleft, V)$ and a state $w \in W$ such that $(W, R, \triangleleft) \in \mathcal{C}$ and $\mathcal{M}, w \models \varphi$. The satisfiability problem of PDDL over a class \mathcal{C} of frames is the decision problem determining whether a PDDL formula is satisfiable in \mathcal{C} .

3 Fischer-Ladner Closure

In [10], Fischer and Ladner proved the strong finite model property of PDL by means of the filtration by a set of formulas called the Fischer-Ladner closure.

To cope with nondeterministic choice and iteration, the original Fischer-Ladner closure extends PDL's language with new propositional variables. In the case of PPDL, a more involved extension of the language is needed to cope with parallel composition of programs. We first introduce this extension before defining the Fischer-Ladner closure adapted to PPDL.

3.1 Placeholders and Marking Functions

In order to decompose formulas of the form $[\alpha \parallel \beta] \varphi$ into subformulas, the language is extended with new atomic formulas called placeholders and parallel composition symbols are distinguished by added indices. Using the same sets Φ_0 and Π_0 of propositional variables and atomic programs, the sets Π_{PH} , Φ_{pure} and Φ_{PH} of *annotated programs*, *pure formulas* and *annotated formulas* respectively, are defined by parallel induction as follows:

$$\begin{aligned} \alpha, \beta &:= a \mid (\alpha ; \beta) \mid (\alpha \cup \beta) \mid \varphi? \mid \alpha^* \mid (\alpha \parallel_i \beta) \\ \varphi &:= p \mid \perp \mid \neg \varphi \mid [\alpha] \varphi \\ \psi &:= \varphi \mid (i, j) \mid \neg \psi \mid [\alpha] \psi \end{aligned}$$

where i ranges over \mathbb{N} and j over $\{1, 2\}$. Moreover, for all $i \in \mathbb{N}$, there must be at most one occurrence of \parallel_i in any annotated program, any pure formula and any annotated formula. The integers below the parallel composition symbols are called *indices*. Formulas of the form (i, j) are called *placeholders*.

To interpret the annotated formulas, if placeholders were simply considered as new propositional variables, it would be impossible to ensure that whenever $w \triangleleft (x, y)$ and $\mathcal{M}, w \vDash [\alpha \parallel_i \beta] \varphi$ then $\mathcal{M}, x \vDash [\alpha](i, 1)$ and $\mathcal{M}, y \vDash [\beta](i, 2)$. Therefore we add flexibility in the interpretation of placeholders by adding *marking functions* which are functions from placeholders to subset of W . The set of all such functions is denoted by B_W . The *empty marking function* $m_W^\emptyset \in B_W$ binds the empty set to all placeholders. The 4-ary forcing relation \vDash_F is defined on all models $\mathcal{M} = (W, R, \triangleleft, V)$, all $w \in W$, all $m \in B_W$ and all $\varphi \in \Phi_{PH}$ by parallel induction along with the extension of R to all annotated programs, in a similar way than for PPDL except:

$$\begin{aligned} \mathcal{M}, x, m \vDash_F (i, j) & \quad \text{iff } x \in m(i, j) \\ x R(\varphi?) y & \quad \text{iff } x = y \text{ and } \mathcal{M}, x, m_W^\emptyset \vDash_F \varphi \\ x R(\alpha \parallel_i \beta) y & \quad \text{iff } \exists w_1, w_2, w_3, w_4, \\ & \quad x \triangleleft (w_1, w_2), w_1 R(\alpha) w_3, w_2 R(\beta) w_4 \text{ and } y \triangleleft (w_3, w_4) \end{aligned}$$

There exists a forgetful surjection $\bar{\cdot} : \Phi_{\text{pure}} \longrightarrow \bar{\Phi}$ associating to each pure formula φ the formula $\bar{\varphi}$ obtained by removing all indices in φ . Thanks to the following lemma, we will consider satisfiability of pure formulas instead of satisfiability of PPDL formulas.

Lemma 1. *For all model $\mathcal{M} = (W, R, \triangleleft, V)$, all $\varphi \in \Phi_{\text{pure}}$ and all $w \in W$, $\mathcal{M}, w, m_W^\emptyset \vDash_F \varphi$ iff $\mathcal{M}, w \vDash \bar{\varphi}$.*

$$\begin{array}{c}
 \frac{(\mu, \varphi)}{(\mu, \neg\varphi)} \\
 \frac{(\mu, [\alpha; \beta] \varphi)}{(\mu, [\alpha][\beta] \varphi)} \\
 \frac{(\mu, [\varphi?] \psi)}{(\mu, \varphi) \quad (\mu, \psi)} \\
 \hline
 \frac{(\mu, [\alpha \parallel_i \beta] \varphi)}{(\mu.L, [\alpha](i, 1)) \quad (\mu.R, [\beta](i, 2)) \quad (\mu, \varphi)}
 \end{array}
 \qquad
 \begin{array}{c}
 \frac{(\mu, [a] \varphi)}{(\mu, \varphi)} \\
 \frac{(\mu, [\alpha \cup \beta] \varphi)}{(\mu, [\alpha] \varphi) \quad (\mu, [\beta] \varphi)} \\
 \frac{(\mu, [\alpha^*] \varphi)}{(\mu, [\alpha][\alpha^*] \varphi) \quad (\mu, \varphi)}
 \end{array}$$

Fig. 1. Fischer-Ladner closure calculus

3.2 Fischer-Ladner Closure

The Fischer-Ladner closure is a decomposition of any PDL formula into a set containing sufficiently many subformulas for the filtration. In the case of PDDL, we need to keep track of the level of separation (called *depth*) of each subformula. Hence we consider *localized formulas*. A *location* is a word on the alphabet $\{L, R\}$, the empty word being denoted by ϵ . A localized formula is a pair (μ, φ) composed of a location μ and a formula φ .

Then, given a localized formula (μ, φ) over Φ_0 and Π_0 , we construct the *closure* $\text{Cl}(\mu, \varphi)$ of (μ, φ) by applying the rules in Fig. 1. In the remainder of this paper we will be mainly interested in closure of localized formulas of the form (ϵ, φ_0) where φ_0 is a pure formula. For all pure formula $\varphi_0 \in \Phi_{\text{pure}}$, we define the abbreviations $\text{FL}(\varphi_0) = \text{Cl}(\epsilon, \varphi_0)$ and $\text{SP}(\varphi_0) = \{\alpha \mid \exists \mu, \exists \varphi, (\mu, \langle \alpha \rangle \varphi) \in \text{FL}(\varphi_0)\}$. The cardinality of $\text{FL}(\varphi_0)$ is denoted by N_{φ_0} . The proof from [10] can be easily adapted to prove the following lemma:

Lemma 2. N_{φ_0} is linear in $|\varphi_0|$.

4 Threads, Twines and Neat Models

In this section, new concepts about PDDL’s models are introduced. These concepts allow us to restrict the class of models to consider for satisfiability. In the next section, we prove that any formula satisfiable in the class of \triangleleft -deterministic models is satisfied in a model with these additional properties. Firstly, to bound the depth of separation of states, we introduce the notion of hierarchical models. This directly corresponds to locations of formulas from the previous section.

Definition 1. Given a model $\mathcal{M} = (W, R, \triangleleft, V)$, a function $\lambda : W \rightarrow \{L, R\}^*$ is a hierarchy function for \mathcal{M} iff

$$\forall x, y, z \in W, \quad x \triangleleft (y, z) \Rightarrow \lambda(y) = \lambda(x).L \text{ and } \lambda(z) = \lambda(x).R \quad (1)$$

$$\forall x, y \in W, \forall \alpha \in \Pi_{PH}, \quad x R(\alpha) y \Rightarrow \lambda(x) = \lambda(y) \quad (2)$$

$\lambda(x)$ is called the depth of x . A model for which there exists a hierarchy function is a hierarchical model.

Secondly, in order to restrict the number of states at each level of separation, the notion of reachability is extended. Given a \triangleleft -deterministic model $\mathcal{M} = (W, R, \triangleleft, V)$, consider the reachability relation $R^\exists = \cup_{\alpha \in \Pi_{PH}} R(\alpha)$. This relation is obviously reflexive. Hence its symmetric and transitive closure, denoted by \sim , is an equivalence relation. The equivalence classes of W by \sim are called *threads* and \sim the thread relation. Notice that if \mathcal{M} is hierarchical, all states in any thread T have the same depth, noted $\lambda(T)$. To strengthen the link between threads and depth, threads are grouped into pairs, each thread of a pair corresponding to one side of the separations. These pairs of threads are called *twines* and are formally defined as follows:

Definition 2. A twine is a pair (T_L, T_R) of threads such that for all $x, y, z \in W$ if $x \triangleleft (y, z)$ then $y \notin T_R, z \notin T_L$ and $y \in T_L \Leftrightarrow z \in T_R$.

In the remainder of this paper, a twine (T_1, T_2) is identified with the set $T_1 \cup T_2$. Obviously, if a thread T is such that for all $(x, y, z) \in \triangleleft, y \notin T$ and $z \notin T$, then for any thread T' having the same property, (T, T') is a twine. Such a thread is called an *isolated thread*. It can be easily proved that if (T_1, T_2) and (T_1, T_3) are twines, then either T_1, T_2 and T_3 are isolated or $T_2 = T_3$. We can now define the notion of neat models.

Definition 3. A model $\mathcal{M} = (W, R, \triangleleft, V)$ is neat if it satisfies all the following conditions:

1. For any thread T_1 there exists a thread T_2 such that (T_1, T_2) or (T_2, T_1) is a twine;
2. There is exactly one isolated thread T_0 ;
3. There exists a hierarchy function λ for \mathcal{M} such that $\lambda(T_0) = \epsilon$.

5 Neat Model Property

In this section we will prove that whenever a pure formula is satisfiable in $\mathcal{C}_{\triangleleft\text{-det}}$, it is satisfiable in a \triangleleft -deterministic neat model. Supposing the pure formula φ_0 is satisfiable, the proof proceeds as follows:

- by Lemma 3 below, there exists a countable model $\mathcal{M}_{\mathfrak{B}}$ satisfying φ_0 ;
- in Sect. 5.1, $\mathcal{M}_{\mathfrak{B}}$ is unraveled into $\mathcal{M}_{\mathfrak{U}}$.
- in Sect. 5.2, unreachable states from $\mathcal{M}_{\mathfrak{U}}$ are pruned to obtain $\mathcal{M}_{\mathfrak{T}}$ and $\mathcal{M}_{\mathfrak{T}}$ is proved to be a \triangleleft -deterministic neat model satisfying φ_0 .

Lemma 3. For any satisfiable pure formula φ_0 , there exists a countable model satisfying φ_0 .

Proof. By a proof similar to Corollary 6.3 in [3]. □

5.1 Unraveling

Let $\mathcal{M}_{\mathfrak{B}} = (W_{\mathfrak{B}}, R_{\mathfrak{B}}, \triangleleft_{\mathfrak{B}}, V_{\mathfrak{B}})$ be a countable \triangleleft -deterministic model satisfying a formula φ_0 at x'_0 . We will construct the *unraveling* of $\mathcal{M}_{\mathfrak{B}}$ at x'_0 . The following method is an adaptation of the well-known unraveling method (see [5] for instance). The key difference is that the resulting model is not a tree-like model.

Let W_{∞} be a countably infinite set. For all $k \in \mathbb{N}$ we will construct the tuple $U_k = (\mathcal{M}_k, h_k)$ such that $\mathcal{M}_k = (W_k, R_k, \triangleleft_k, V_k)$ is a model with $W_k \subseteq W_{\infty}$ and h_k is a homomorphism from W_k to $W_{\mathfrak{B}}$, thus preserving valuation. The initial tuple U_0 is such that $W_0 = \{x_0\}$ for some $x_0 \in W_{\infty}$, $R_0(a) = \emptyset$ for all $a \in \Pi_0$, $\triangleleft_0 = \emptyset$ and $h_0(x_0) = x'_0$. Then for all $k \in \mathbb{N}$, U_{k+1} is constructed from U_k by fixing one of the following defects for some $v, w_1, w_2 \in W_k$, $a \in \Pi_0$ and $w', w'_1, w'_2 \in W_{\mathfrak{B}}$:

Successor Defect (v, a, w') . If $h_k(v) R_{\mathfrak{B}}(a) w'$ but there is no $w \in W_k$ such that $h_k(w) = w'$ and $v R_k(a) w$, then U_{k+1} is obtained from U_k by adding a new state $w \in W_{\infty} \setminus W_k$ such that $h_{k+1}(w) = w'$ and $v R_{k+1}(a) w$,

Split Defect (v, w'_1, w'_2) . If $h_k(v) \triangleleft_{\mathfrak{B}} (w'_1, w'_2)$ but there are no $w_1, w_2 \in W_k$ such that $h_k(w_1) = w'_1$, $h_k(w_2) = w'_2$ and $v \triangleleft_k (w_1, w_2)$, then U_{k+1} is obtained from U_k by adding two new states $w_1, w_2 \in W_{\infty} \setminus W_k$ such that $h_{k+1}(w_1) = w'_1$, $h_{k+1}(w_2) = w'_2$ and $v \triangleleft_{k+1} (w_1, w_2)$.

Merge Defect (w', w_1, w_2) . If $w' \triangleleft_{\mathfrak{B}} (h_k(w_1), h_k(w_2))$ but there is no $w \in W_k$ such that $h_k(w) = w'$ and $w \triangleleft_k (w_1, w_2)$, then U_{k+1} is obtained from U_k by adding a new state $w \in W_{\infty} \setminus W_k$ such that $h_{k+1}(w) = w'$ and $w \triangleleft_{k+1} (w_1, w_2)$.

Since W_{∞} , Π_0 and $W_{\mathfrak{B}}$ are countable sets, there is a sequence $\delta_0, \delta_1, \dots$ of possible defects such that each possible defect appears infinitely often. We enforce that for all $k \in \mathbb{N}$, either δ_k is a defect for U_k fixed in U_{k+1} or δ_k is not a defect for U_k and $U_{k+1} = U_k$. The unraveling $\mathcal{M}_{\mathfrak{U}} = (W_{\mathfrak{U}}, R_{\mathfrak{U}}, \triangleleft_{\mathfrak{U}}, V_{\mathfrak{U}})$ of $\mathcal{M}_{\mathfrak{B}}$ at x'_0 is the union of \mathcal{M}_k for all $k \in \mathbb{N}$.

Proposition 1. $\mathcal{M}_{\mathfrak{U}}$ is a \triangleleft -deterministic model satisfying φ_0 .

To prove Proposition 1, we adapt the bounded morphism definition to PPDL and prove Lemma 4.

Definition 4. Given two \triangleleft -deterministic models $\mathcal{M} = (W, R, \triangleleft, V)$ and $\mathcal{M}' = (W', R', \triangleleft', V')$, a mapping $h : \mathcal{M} \rightarrow \mathcal{M}'$ is a bounded morphism iff it satisfies the following conditions for all $v, w, w_1, w_2 \in W$, $w', w'_1, w'_2 \in W'$ and $a \in \Pi_0$:

$$w \text{ and } h(w) \text{ satisfy the same propositional variables} \quad (3)$$

$$v R(a) w \Rightarrow h(v) R'(a) h(w) \quad (4)$$

$$h(v) R'(a) w' \Rightarrow \exists w, h(w) = w' \text{ and } v R(a) w \quad (5)$$

$$w \triangleleft (w_1, w_2) \Rightarrow h(w) \triangleleft' (h(w_1), h(w_2)) \quad (6)$$

$$h(w) \triangleleft' (w'_1, w'_2) \Rightarrow \exists w_1, w_2, \begin{cases} h(w_1) = w'_1, h(w_2) = w'_2 \\ \text{and } w \triangleleft (w_1, w_2) \end{cases} \quad (7)$$

$$w' \triangleleft' (h(w_1), h(w_2)) \Rightarrow \exists w, h(w) = w' \text{ and } w \triangleleft (w_1, w_2) \quad (8)$$

Lemma 4. *If h is a bounded morphism from \mathcal{M} to \mathcal{M}' , then for all $w \in W$ and $\varphi \in \Phi$, $\mathcal{M}, w \models \varphi$ iff $\mathcal{M}', h(w) \models \varphi$.*

Considering the functions $(h_k)_{k \in \mathbb{N}}$ as subsets of $W_{\mathcal{M}} \times W_{\mathfrak{B}}$, we define h as their union. We prove that h is a bounded morphism, the successor, split and merge defects ensuring conditions (5), (7) and (8) respectively. Finally, since bounded morphisms preserve \triangleleft -determinism, Proposition 1 is proved. Despite $\mathcal{M}_{\mathcal{M}}$ not being tree-like, it has the following form of acyclicity:

Proposition 2. *For all $x, y \in W_{\mathcal{M}}$ and all $\alpha, \beta \in \Pi_{PH}$, if $x R(\alpha) y$ and $y R(\beta) x$ then $x = y$.*

5.2 Pruning

In this section, we remove unreachable states from $\mathcal{M}_{\mathcal{M}}$ and prove that the resulting model is neat. The method consists in identifying reachable threads and relies on the fact that new reachable threads are added only by split defects. We use a function r associating to each state $x \in W_{\mathcal{M}}$ either the first state of x 's thread if this thread is reachable or the special value **Out** otherwise. The function $r : W_{\mathcal{M}} \longrightarrow W_{\mathcal{M}} \cup \{\text{Out}\}$ is formally defined by induction on the construction of $\mathcal{M}_{\mathcal{M}}$ as follows:

0. Initially, $r(x_0) = x_0$;
1. When fixing a successor defect (w, a, v) by adding w' , $r(w') = r(w)$;
2. When fixing a split defect (w, v_1, v_2) by adding w_1 and w_2 , if $r(w) \neq \text{Out}$ then $r(w_1) = w_1$ and $r(w_2) = w_2$, otherwise $r(w_1) = r(w_2) = \text{Out}$;
3. When fixing a merge defect $\delta_k = (v, w_1, w_2)$ by adding w , if there exists $w' \in W_k$ such that $w' \triangleleft_k (r(w_1), r(w_2))$ then $r(w) = r(w')$, otherwise $r(w) = \text{Out}$.

The function r is well-defined because $\mathcal{M}_{\mathcal{M}}$ is \triangleleft -deterministic. Then, the model $\mathcal{M}_{\mathfrak{N}} = (W_{\mathfrak{N}}, R_{\mathfrak{N}}, \triangleleft_{\mathfrak{N}}, V_{\mathfrak{N}})$ is defined as the reduction of $\mathcal{M}_{\mathcal{M}}$ to the worlds x for which $r(x) \neq \text{Out}$. The following proposition can easily be proved:

Proposition 3. *$\mathcal{M}_{\mathfrak{N}}$ is a \triangleleft -deterministic model satisfying φ_0 at x_0 .*

It remains to prove that $\mathcal{M}_{\mathfrak{N}}$ is neat. Let $\sim_{\mathfrak{N}}$ be the thread relation of $\mathcal{M}_{\mathfrak{N}}$. The proof of Proposition 4 relies on the following two lemmas:

Lemma 5. *For all $x, y \in W_{\mathfrak{N}}$, $r(x) = r(y)$ iff $x \sim_{\mathfrak{N}} y$.*

Lemma 6. *If $z \triangleleft_{\mathfrak{N}} (x, y)$ then there exists $z' \in W_{\mathfrak{N}}$ such that $z' \triangleleft_{\mathfrak{N}} (r(x), r(y))$ and $(z', r(x), r(y))$ has been added to $\triangleleft_{\mathcal{M}}$ by a split defect.*

Proposition 4. *$\mathcal{M}_{\mathfrak{N}}$ is neat.*

Proof sketch. For the first two conditions of Definition 3, we prove the corresponding two properties using Lemma 6:

1. For all $x_1, x_2, y_1, y_2, z_1, z_2 \in W_{\mathfrak{N}}$, if $z_1 \triangleleft_{\mathfrak{N}} (x_1, y_1)$ and $z_2 \triangleleft_{\mathfrak{N}} (x_2, y_2)$ then $r(x_1) = r(x_2) \Leftrightarrow r(y_1) = r(y_2)$.

2. x_0 is the only $x \in W_{\mathfrak{N}}$ such that $r(x) = x$ and for all $(w, y, z) \in \triangleleft_{\mathfrak{N}}$, $r(y) \neq x$ and $r(z) \neq x$.

For the last condition of Definition 3, the hierarchy function λ is constructed such that:

- $\lambda(x_0) = \epsilon$;
- for any split defect (w, v_1, v_2) adding w_1 and w_2 to $W_{\mathfrak{M}}$, if $r(w) \neq \text{Out}$ then $\lambda(w_1) = \lambda(w).L$ and $\lambda(w_2) = \lambda(w).R$;
- for all x , $\lambda(x) = \lambda(r(x))$. □

6 Piecewise Filtration

In this section, we prove the following proposition:

Proposition 5. *Whenever a formula $\varphi \in \Phi$ is satisfiable in a \triangleleft -deterministic neat model, it is satisfiable in a \triangleleft -deterministic finite model $\mathcal{M} = (W, R, \triangleleft, V)$ in which the cardinality of W is bounded by an exponential in the number of symbols in φ .*

Suppose $\mathcal{M}_{\mathfrak{N}} = (W_{\mathfrak{N}}, R_{\mathfrak{N}}, \triangleleft_{\mathfrak{N}}, V_{\mathfrak{N}})$ is neat and $\mathcal{M}_{\mathfrak{N}}, x_0, m_W^{\emptyset} \models_{\mathbb{F}} \varphi_0$ for some $x_0 \in W_{\mathfrak{N}}$ and $\varphi_0 \in \Phi_{\text{pure}}$. Furthermore, we suppose λ is a hierarchical function for $\mathcal{M}_{\mathfrak{N}}$ such that $\lambda(x_0) = \epsilon$. The model $\mathcal{M}_{\mathfrak{F}}$ satisfying Proposition 5 is inductively constructed from $\mathcal{M}_{\mathfrak{N}}$. At the initial step, the filtration of the thread containing x_0 is added to $\mathcal{M}_{\mathfrak{F}}$. At the inductive steps, for each pair of states in $\mathcal{M}_{\mathfrak{F}}$ which must be connected by a parallel program, the filtration of a twine of $\mathcal{M}_{\mathfrak{N}}$ corresponding to this parallel program is added to $\mathcal{M}_{\mathfrak{F}}$.

In order to preserve the \triangleleft -determinism of $\mathcal{M}_{\mathfrak{N}}$ during the filtration, we need to distinguish for any filtered twine, the forward (split) decomposition from the backward (merge) one. For that matter, placeholders are duplicated and the special pair $\{(0, 1), (0, 2)\}$ of placeholders is used to mark the forward decomposition. Formally, for any formula $\varphi \in \Phi_{PH}$ and any $k \in \mathbb{N}$, let $f_{\text{dup}}(k, \varphi)$ be the formula obtained from φ by replacing each occurrence of (i, j) in φ by $(2i + k, j)$, for all $i \in \mathbb{N}$ and $j \in \{1, 2\}$. We define the sets

$$\begin{aligned} \text{FL}^+(\varphi_0) &= \{(\mu, f_{\text{dup}}(k, \varphi)) \mid k \in \{1, 2\}, (\mu, \varphi) \in \text{FL}(\varphi_0)\} \cup \\ &\quad \{((\mu, (0, j)), (\mu, \neg(0, j))) \mid j \in \{1, 2\} \text{ and } \exists \varphi, (\mu, \varphi) \in \text{FL}(\varphi_0)\} \\ \text{SF}^+(\varphi_0) &= \{\varphi \mid \exists \mu, (\mu, \varphi) \in \text{FL}^+(\varphi_0)\} \end{aligned}$$

The filtrations are done using the \equiv_m equivalence relations over $W_{\mathfrak{N}}$, defined for any marking function $m \in B_{W_{\mathfrak{N}}}$ such that $x \equiv_m y$ iff $\lambda(x) = \lambda(y)$ and for all $(\mu, \varphi) \in \text{FL}^+(\varphi_0)$ if $\mu = \lambda(x)$ then $\mathcal{M}_{\mathfrak{N}}, x, m \models_{\mathbb{F}} \varphi \Leftrightarrow \mathcal{M}_{\mathfrak{N}}, y, m \models_{\mathbb{F}} \varphi$. The functions Ω and Ψ are defined for all $X \subseteq W_{\mathfrak{N}}$ and $m \in B_{W_{\mathfrak{N}}}$ by:

$$\begin{aligned} \Omega(X, m) &= \{Y \cap X \mid Y \in W_{\mathfrak{N}}/\equiv_m\} \\ \Psi(X, m) &= \{\varphi \mid \exists x \in W_{\mathfrak{N}}, (\lambda(x), \varphi) \in \text{FL}^+(\varphi_0) \text{ and } \mathcal{M}_{\mathfrak{N}}, x, m \models_{\mathbb{F}} \varphi\} \end{aligned}$$

Finally, the set PF references all the parallel program links for which we may have to add the filtration of a twine. Formally, PF is the greatest subset of $\mathbb{N} \times \mathcal{P}(\text{SF}^+(\varphi_0)) \times \text{SP}(\varphi_0) \times \mathcal{P}(\text{SF}^+(\varphi_0))$ such that for all $(k, F, \alpha, G) \in \text{PF}$, α is of the form $(\alpha_1 \parallel_i \alpha_2)$ and there exists $\mu \in \{L, R\}^*$ and $\varphi \in \Phi_{PH}$ such that $(\mu, \langle \alpha \rangle \varphi) \in \text{FL}^+(\varphi_0)$ and for all $\psi \in F \cup G$, $(\mu, \psi) \in \text{FL}^+(\varphi_0)$. Since $\mathcal{P}(\text{SF}^+(\varphi_0))$ and $\text{SP}(\varphi_0)$ are finite, there exists a total order over PF with a least element and such that $(k, F, \alpha, G) < (k', F', \alpha', G')$ implies $k \leq k'$. This order determines a bijective function from \mathbb{N} to PF. Moreover, if (k, F, α, G) is the n^{th} tuple in PF then $k \leq n$.

Now we inductively construct the models $\mathcal{M}_n = (W_n, R_n, \triangleleft_n, V_n)$ for $n \in \mathbb{N}$, where $W_n \subseteq \mathbb{N} \times \mathcal{P}(W_{\mathfrak{N}}) \times B_{W_{\mathfrak{N}}}$. The following invariants hold for all $n \in \mathbb{N}$:

- for all $(k, X, m) \in W_n$, all $\varphi \in \Psi(X, m)$ and all $x \in X$, $\mathcal{M}_{\mathfrak{N}}, x, m \models_F \varphi$;
- for all $(k, X, m), (k', Y, m') \in W_n$, if $k = k'$ then $m = m'$ and for all $x \in X$ and all $y \in Y$, x and y belong to the same twine and if $x \equiv_m y$ then $X = Y$.

Initial step. Let T_0 be the thread in $\mathcal{M}_{\mathfrak{N}}$ containing x_0 . We set:

$$\begin{aligned} W_0 &= \{(0, X, m_{W_{\mathfrak{N}}}^{\emptyset}) \mid X \in \Omega(T_0, m_{W_{\mathfrak{N}}}^{\emptyset})\} \\ R_0(a) &= \{((k, X, m), (k', X', m')) \in W_0 \times W_0 \mid \\ &\quad k = k' \text{ and } \exists x \in X, \exists x' \in X', x R_{\mathfrak{N}}(a) x'\} \\ \triangleleft_0 &= \emptyset \\ V_0(p) &= \{(k, X, m) \in W_0 \mid p \in \Psi(X, m)\} \end{aligned}$$

If $\text{PF} = \emptyset$ then $\mathcal{M}_n = \mathcal{M}_0$ for all $n \in \mathbb{N}$. Otherwise the following inductive step is applied.

Inductive step. Suppose \mathcal{M}_n has already been defined and let $(k, F, \alpha_1 \parallel_i \alpha_2, G)$ be the n^{th} tuple in PF. If for all $X, Y \subseteq W_{\mathfrak{N}}$ and all $m \in B_{W_{\mathfrak{N}}}$, one of the following conditions is not satisfied

$$\Psi(X, m) = F \text{ and } \Psi(Y, m) = G \quad (9)$$

$$(k, X, m) \in W_n \text{ and } (k, Y, m) \in W_n \quad (10)$$

$$\exists x \in X, \exists y \in Y, x R_{\mathfrak{N}}(\alpha_1 \parallel_i \alpha_2) y \quad (11)$$

then $\mathcal{M}_{n+1} = \mathcal{M}_n$. Otherwise, by the invariants, there is exactly one tuple (X, Y, m) satisfying (9) and (10). By condition (11), there exists $x \in X$, $y \in Y$ and $w_1, w_2, w_3, w_4 \in W_{\mathfrak{N}}$ such that $x \triangleleft_{\mathfrak{N}}(w_1, w_2)$, $w_1 R_{\mathfrak{N}}(\alpha_1) w_3$, $w_2 R_{\mathfrak{N}}(\alpha_2) w_4$ and $y \triangleleft_{\mathfrak{N}}(w_3, w_4)$. The marking function m_{n+1} is defined such that

- $m_{n+1}(0, j) = \{w_j\}$;
- $m_{n+1}(i, j) = \left\{ w \mid \exists \beta_1, \beta_2. (\beta_1 \parallel_{\frac{i-1}{2}} \beta_2) \in \text{SP}(\varphi_0) \text{ and } w_j R_{\mathfrak{N}}(\beta_j) w \right\}$ if i is odd;
- $m_{n+1}(i, j) = \left\{ w \mid \exists \beta_1, \beta_2. (\beta_1 \parallel_{\frac{i-2}{2}} \beta_2) \in \text{SP}(\varphi_0) \text{ and } w_{j+2} R_{\mathfrak{N}}(\beta_j) w \right\}$ if i is even and positive.

Since $\mathcal{M}_{\mathfrak{N}}$ is neat, w_1, w_2, w_3 and w_4 belong to the same twine θ . For all $t \in 1..4$, there exists $X_t \in \Omega(\theta, m_{n+1})$ such that $x_t \in X_t$. \mathcal{M}_{n+1} is defined by:

$$\begin{aligned} W_{n+1} &= W_n \cup \{(n+1, X, m_{n+1}) \mid X \in \Omega(\theta, m_{n+1})\} \\ R_{n+1}(a) &= \{((k, X, m), (k', X', m')) \in W_{n+1} \times W_{n+1} \mid \\ &\quad k = k' \text{ and } \exists x \in X, \exists x' \in X', x R_{\mathfrak{N}}(a) x'\} \\ \triangleleft_{n+1} &= \triangleleft_n \cup \{((k, X, m), (n+1, X_1, m_{n+1}), (n+1, X_2, m_{n+1})), \\ &\quad ((k, Y, m), (n+1, X_3, m_{n+1}), (n+1, X_4, m_{n+1}))\} \\ V_{n+1}(p) &= \{(k, X, m) \in W_{n+1} \mid p \in \Psi(X, m)\} \end{aligned}$$

Finally, $\mathcal{M}_{\mathfrak{F}} = (W_{\mathfrak{F}}, R_{\mathfrak{F}}, \triangleleft_{\mathfrak{F}}, V_{\mathfrak{F}})$ is defined as the union of \mathcal{M}_n for all $n \in \mathbb{N}$ and we prove that $\mathcal{M}_{\mathfrak{F}}$ satisfies Proposition 5.

Proof sketch of Proposition 5. To prove that the cardinality of $W_{\mathfrak{F}}$ is bounded by an exponential in $|\varphi_0|$, we consider the graph whose vertices are sets of $W_{\mathfrak{F}}$'s states having the same first component and such that there is an edge from G to G' iff states in G' have been added to $\mathcal{M}_{\mathfrak{F}}$ to connect two states in G . We prove that each vertex contains an exponential number of states and that the graph is a tree with branching factor exponential in $|\varphi_0|$ and depth linear in $|\varphi_0|$. \triangleleft -determinism of $\mathcal{M}_{\mathfrak{F}}$ is ensured by the interpretation of the placeholders (0, 1) and (0, 2). Finally, to prove the truth lemma, we prove the following properties by simultaneous induction on $|\varphi|$ for 1 and on $|\alpha|$ for 2 and 3:

1. For all $(k, X, m) \in W_{\mathfrak{F}}$ and all formula φ such that $(\lambda(X), \varphi) \in \text{FL}^+(\varphi_0)$, $\varphi \in \Psi(X, m) \Leftrightarrow \mathcal{M}_{\mathfrak{F}}, (k, X, m), m_{\mathfrak{F}} \models_{\text{F}} \varphi$.
2. If $(k, X, m) \in W_{\mathfrak{F}}$, $x \in X$, θ_x is the twine of x , $Y \in \Omega(\theta_x, m)$, $y \in Y$, $(\lambda(x), \langle \alpha \rangle \varphi) \in \text{FL}(\varphi_0)$ and $x R_{\mathfrak{N}}(\alpha) y$, then $(k, X, m) R_{\mathfrak{F}}(\alpha) (k, Y, m)$.
3. If $(k, X, m) R_{\mathfrak{F}}(\alpha) (k, Y, m)$ and $[\alpha]\varphi \in \Psi(X, m)$, then $\varphi \in \Psi(Y, m)$. \square

Finally, since the model checking problem for PDDL is obviously polynomial in the number of states of the model, we deduce a complexity upper bound:

Proposition 6. *The satisfiability problem of PDDL interpreted over \triangleleft -deterministic frames is in NEXPTIME.*

7 Conclusion

In this paper, we prove that PDDL interpreted over the class $\mathcal{C}_{\triangleleft\text{-det}}$ of \triangleleft -deterministic frames has a strong finite model property and that the satisfiability problem of this logic is in NEXPTIME. These results rely on the neat model property introduced in the paper and are obtained by a piecewise filtration using an adaptation of the Fischer-Ladner closure. Because formulas with parallel compositions cannot be properly decomposed into subformulas, the language is extended with indices and placeholders. We hope these new concepts will be useful in future works. We briefly list some possibilities. First, a tight complexity

result for PDDL over $\mathcal{C}_{<-\text{det}}$ remains to be found. Secondly, the complexity of PDDL interpreted over the class of neat frames could be studied. Finally, since no semantic equivalents of the multiplicative implication of BBI can be defined in PDDL over $\mathcal{C}_{<-\text{det}}$, it could explicitly be added to the language.

References

1. Abrahamson, K.R.: Modal logic of concurrent nondeterministic programs. In: Kahn, G. (ed.) *Semantics of Concurrent Computation*. LNCS, vol. 70, pp. 21–33. Springer, Heidelberg (1979)
2. Balbiani, P., Boudou, J.: Iteration-free PDL with storing, recovering and parallel composition: a complete axiomatization. *J. Logic Comput.* (2015, to appear)
3. Balbiani, P., Tinchev, T.: Definability and computability for PRSPDL. In: *Advances in Modal Logic*, pp. 16–33. College Publications (2014)
4. Benevides, M.R.F., de Freitas, R.P., Viana, J.P.: Propositional dynamic logic with storing, recovering and parallel composition. *ENTCS* **269**, 95–107 (2011)
5. Blackburn, P., de Rijke, M., Venema, Y.: *Modal Logic*. Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press, Cambridge (2001)
6. Cardelli, L., Gordon, A.D.: Anytime, anywhere: modal logics for mobile ambients. In: *POPL*, pp. 365–377. ACM (2000)
7. Collinson, M., Pym, D.J.: Algebra and logic for resource-based systems modelling. *Math. Struct. Comput. Sci.* **19**(5), 959–1027 (2009)
8. Demri, S., Deters, M.: Separation logics and modalities: a survey. *J. Appl. Non Class. Logics* **25**(1), 50–99 (2015)
9. van Ditmarsch, H., van der Hoek, W., Kooi, B.P.: *Dynamic Epistemic Logic*, vol. 337. Springer Science and Business Media, Heidelberg (2007)
10. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.* **18**(2), 194–211 (1979)
11. Harel, D.: Recurring dominoes: making the highly undecidable highly understandable (preliminary report). In: Budach, L. (ed.) *Fundamentals of Computation Theory*. LNCS, vol. 158, pp. 177–194. Springer, Heidelberg (1983)
12. Lange, M., Lutz, C.: 2-exptime lower bounds for propositional dynamic logics with intersection. *J. Symb. Log.* **70**(4), 1072–1086 (2005)
13. Larchey-Wendling, D., Galmiche, D.: The undecidability of boolean BI through phase semantics. In: *LICS*, pp. 140–149. IEEE Computer Society (2010)
14. Marx, M., Pólos, L., Masuch, M.: *Arrow Logic and Multi-modal Logic*. CSLI Publications, Stanford (1996)
15. Mayer, A.J., Stockmeyer, L.J.: The complexity of PDL with interleaving. *Theor. Comput. Sci.* **161**(1–2), 109–122 (1996)
16. Peleg, D.: Concurrent dynamic logic. *J. ACM* **34**(2), 450–479 (1987)
17. Pym, D.J.: *The semantics and proof theory of the logic of bunched implications*, Applied Logic Series, vol. 26. Kluwer Academic Publishers (2002)
18. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: *LICS*. pp. 55–74. IEEE Computer Society (2002)

A Circuit Complexity Approach to Transductions

Michaël Cadilhac¹(✉), Andreas Krebs¹,
Michael Ludwig¹, and Charles Paperman²

¹ Wilhelm Schickard Institut, Universität Tübingen, Tübingen, Germany
michael@cadilhac.name, mail@krebs-net.de,
ludwigm@informatik.uni-tuebingen.de

² University of Warsaw and Warsaw Center of Mathematics
and Computer Science, Warsaw, Poland
Charles.Paperman@liafa.univ-paris-diderot.fr

Abstract. Low circuit complexity classes and regular languages exhibit very tight interactions that shade light on their respective expressiveness. We propose to study these interactions at a functional level, by investigating the deterministic rational transductions computable by constant-depth, polysize circuits. To this end, a circuit framework of independent interest that allows variable output length is introduced. Relying on it, there is a general characterization of the set of transductions realizable by circuits. It is then decidable whether a transduction is definable in AC^0 and, assuming a well-established conjecture, the same for ACC^0 .

Introduction

The regular languages in circuit complexity classes play an instrumental role in some of the most emblematic results of circuit complexity. The celebrated result of Furst, Saxe and Sipser [11] shows that the regular language $\text{PARITY} = \{w \in \{0,1\}^* \mid |w|_1 \equiv 0 \pmod{2}\}$ is not in AC^0 , the class of constant-depth, polysize, unbounded fan-in circuits. As PARITY belongs to ACC^0 (which allows in addition unbounded fan-in modulo gates), this separates AC^0 and ACC^0 . Barrington's theorem [1] states that the regular languages are complete for the class NC^1 of logdepth, polysize, and constant fan-in circuits. Further, Koucký, Pudlák, and Thérien [12] show that regular languages separate classes defined by ACC^0 circuits using linear number of gates and using linear number of wires.

The classification of regular languages within circuit complexity classes thus attracted interest, culminating in the results of Barrington *et al.* [2] that entirely describe the regular languages in AC^0 , ACC^0 and NC^1 . The algebraic property of regular languages studied therein deviates sharply from the prevailing line of work at the time, which relied on the study of the syntactic monoids of regular languages. (The syntactic monoid is the monoid of transformations of states of the minimal automaton.) Indeed, $\text{PARITY} \notin AC^0$, while the language EVEN of even-length words over $\{0,1\}$, which has the same syntactic monoid, does

belong to AC^0 . Hence the class of regular languages in AC^0 does not admit a characterization solely in terms of the syntactic monoids.

We propose to take this study to the functional case, that is, to characterize the functions realized by rational transducers (i.e., input/output automata) that are expressible by an AC^0 circuit family. Similarly to the context at the time of [2], we face a situation where, to the best of our knowledge, most characterizations focused on algebraic properties that would blur the line between PARITY and EVEN (e.g., [14]).

We rely on a property we call *continuity* for a class of languages \mathcal{V} , as borrowed from the field of topology: a transduction τ is \mathcal{V} -continuous if it preserves \mathcal{V} by inverse image (i.e., $\forall L \in \mathcal{V}, \tau^{-1}(L) \in \mathcal{V}$). It is well known that any transduction τ is continuous for the regular languages; together with an additional property on the output length of τ , this even characterizes deterministic transductions [3]. Namely, with $d(u, v) = |u| + |v| - |u \wedge v|$, where $u \wedge v$ is the largest common prefix of u and v , the latter property is that $d(\tau(u), \tau(v)) \leq k \times d(u, v)$, a strong form of *uniform* continuity. Continuity thus appears as a natural invariant when characterizing transductions—the *forward* behaviors of τ , that is, its images, are less relevant, as any NP problem is the image of Σ^* under an AC^0 function [4]. Our contributions are three-fold:

- We propose a model of circuits that allows for functions of unrestricted output length: as opposed to previous models, e.g., [19], we do not impose the existence of a mapping between the input and output lengths.
- Relying on this model, we characterize the deterministic rational transductions computed by AC^0 circuits with access to gates in a given class. This characterization relies for one part on algebraic objects similar to the ones used in [2], through the use of the modern framework of *lm-varieties* [18]. For the other part, we rely on the notion of *continuity*. This bears a striking resemblance to the characterization of Reutenauer and Schützenberger [16] of the transductions with a group as transition monoid.
- The characterization then leads to the decidability of the membership of a deterministic rational transduction in AC^0 or in ACC^0 . This is effective in the sense that an appropriate circuit can be produced realizing the transduction.

In Sect. 1, we succinctly cover the automata- and circuit-theoretic notions necessary to our presentation. In Sect. 2, we introduce the circuit model for variable output length functions and argue for its legitimacy. In Sect. 3, we show that studying the transition morphism of an automaton is equivalent to studying the languages accepted at each of its states; this enables us to keep to a minimum the algebraic references throughout our presentation. In Sect. 4, we show the aforementioned characterization and delay to Sect. 5 its implications on AC^0 and ACC^0 . We discuss the results and their limitations in Sect. 6.

1 Preliminaries

Monoid, morphisms, quotient. A monoid is a set equipped with a binary associative operation, denoted multiplicatively, with a unit element. For an alphabet Σ ,

the set Σ^* is the free monoid generated by Σ , its unit element being the empty word ε . A morphism is a map $\varphi: M \rightarrow N$ satisfying $\varphi(ab) = \varphi(a)\varphi(b)$ and $\varphi(1) = 1$, with $a, b \in M$ and 1 denoting the unit element of M and N . A morphism $\varphi: \Sigma^* \rightarrow T^*$ is an *lm-morphism*, where lm stands for *length-multiplying*, if there is a k such that $\varphi(\Sigma) \subseteq T^k$. Given a language L and a word u , the *left quotient* of L by u is the set $u^{-1}L = \{v \mid uv \in L\}$. The *right quotient* Lu^{-1} is defined symmetrically. For $w \in \Sigma^*$ and $a \in \Sigma$, we let $|w|_a$ be the number of a 's in w , i.e., the image of w under the morphism $a \mapsto 1, \Sigma \setminus \{a\} \mapsto 0$ into $(\mathbb{N}, +)$.

Circuits. We use standard notations, as presented for instance in [17] and [19]. By AC^0 , we denote the class of languages recognized by constant-depth, polysize circuit families with Boolean gates of unbounded fan-in. We consider *nonuniform* families, that is, we leave unconstrained the mapping from the input size n to the circuit with n inputs. Such families recognize languages in $L \subseteq \{0, 1\}^*$; to extend this to any alphabet Σ , we always assume there is a canonical map from Σ to $\{0, 1\}^{|\Sigma|}$, that lets us encode and decode words of Σ^* in binary. A language L naturally defines an L -gate which outputs 1 iff its input is in L ; for instance, $\{0, 1\}^*1\{0, 1\}^*$ defines the OR gate. For a class of languages \mathcal{V} , we write $AC^0(\mathcal{V})$ for languages recognized by AC^0 circuit families with access to L -gates for all $L \in \mathcal{V}$. We let $ACC^0 = AC^0(\text{MOD})$ where MOD is the class of regular languages on $\{0, 1\}^*$ of the form $\{|w|_1 \equiv 0 \pmod k\}$ for some k . Further, we define $TC^0 = AC^0(\text{MAJ})$ where MAJ is the nonregular language $\{|w|_1 \geq |w|_0 \mid w \in \{0, 1\}^*\}$. We will occasionally rely on the conjectured and widely-believed separation of ACC^0 and TC^0 . Extending circuits to functions, a function f is in FAC^0 if there is a family of constant-depth, polysize circuits with multiple ordered output bits, such that $f(u)$ is the output of the circuit for input size $|u|$. We naturally extend the notation $AC^0(\mathcal{V})$ to $FAC^0(\mathcal{V})$.

Automata. A *deterministic automaton* is a tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite set of states, Σ the alphabet, $\delta: Q \times \Sigma \rightarrow Q$ is a partial transition function, q_0 is the initial state, and F is the set of final states. We naturally extend δ to words by letting $\delta(q, \varepsilon) = q$, and $\delta(q, aw) = \delta(\delta(q, a), w)$ when $\delta(q, a)$ is defined. We always assume that any state q is accessible and coaccessible, i.e., there is a word uv such that $\delta(q_0, u) = q$ and $\delta(q, v) \in F$. We write $L(A, q)$ for $\{w \mid \delta(q_0, w) = q\}$, and $L(A) = \cup_{f \in F} L(A, f)$ for the language of A . For two states q, q' , we say that q can be *separated* from q' in \mathcal{V} if there is a language L in \mathcal{V} such that $L(A, q) \subseteq L \subseteq \overline{L(A, q')}$. An automaton is *all-separable* in \mathcal{V} if each pair of distinct states can be separated in \mathcal{V} . It is *all-definable* in \mathcal{V} if every language $L(A, q)$ is in \mathcal{V} . We often use the shorter terms \mathcal{V} -all-separable and \mathcal{V} -all-definable, of self-explanatory meanings. We write REG for the class of regular languages.

Continuity, Lm-varieties. A mapping $f: \Sigma^* \rightarrow T^*$ is *continuous* for \mathcal{V} , in short \mathcal{V} -*continuous*, if $L \in \mathcal{V}$ implies $f^{-1}(L) \in \mathcal{V}$ —this name stems from the notion of continuity in topology. The sets of regular languages recognized by circuit families form a backbone of our work. It is thus natural to assume that these

sets be closed under operations that AC^0 circuits can compute; this is formalized as follows. A class of languages \mathcal{V} is an *lm-variety* if it is a Boolean algebra of languages closed under left and right quotient such that any lm-morphism is \mathcal{V} -continuous. It can be shown that if $\text{AC}^0(\mathcal{V}) \cap \text{REG} = \mathcal{V}$, then \mathcal{V} is an lm-variety. As is customary, we write \mathcal{QA} for $\text{AC}^0 \cap \text{REG}$ and \mathcal{M}_{sol} for $\text{ACC}^0 \cap \text{REG}$ —these names stem from the algebraic classes recognizing the languages: quasi-a-periodic stamps and solvable monoids respectively, see Sect. 3 and [17] for more details. In particular, $\text{ACC}^0 \neq \text{TC}^0$ iff $\text{AC}^0(\mathcal{M}_{\text{sol}}) \cap \text{REG} = \mathcal{M}_{\text{sol}}$ [2]. In the sequel, the symbol \mathcal{V} always denotes some lm-variety of languages.

Transducers. A *deterministic transducer* is a tuple $A = (Q, \Sigma, T, \delta, \nu, q_0, F)$ which is an automaton equipped with an additional alphabet T and a mapping $\nu: Q \times \Sigma \rightarrow T^*$ of same domain as δ . We extend ν to words in Σ^* by letting $\nu(q, \varepsilon) = \varepsilon$ and $\nu(q, aw) = \nu(q, a)\nu(\delta(q, a), w)$, when $\delta(q, a)$ is defined. The partial function $\tau: \Sigma^* \rightarrow T^*$ mapping $w \in L(A)$ to $\nu(q_0, w)$ is called a *transduction*. A transducer is said to be *output-minimal* if for every pair of states q, q' , there is a word w such that either only one of $\delta(q, w)$ or $\delta(q', w)$ is final, or both are and $\nu(q, w) \neq \nu(q', w)$. For any transduction τ , we fix an arbitrary output-minimal transducer $\text{MinT}(\tau)$ realizing it. Note that given a transducer, one can easily compute an output-minimal transducer realizing the same transduction. We will see that the choice of $\text{MinT}(\tau)$ does not bear any impact on the results.

We freely use Q, Σ, T , etc. when an automaton or a transducer is under study, with the understanding that they are the relevant components of its defining tuple. Our focus being solely on automata, transducers, and transductions that are deterministic, we will omit mentioning determinism from now on.

2 Circuit Frameworks for Variable-Length Functions

In the literature, most of the work on functions computed by circuits focus on variants of the class FAC^0 (see, e.g., [19]). In these, multiple (ordered) output gates are provided, and there is thus an implicit mapping from input length to output length. Towards circumventing this limitation, we propose a few different frameworks, and establish some formal shortcomings in order to legitimize our final choice. Our main requirement is that functions defined using constant-depth, polysize circuits should be AC^0 -continuous—this corresponds to a simple composition of the circuits. In particular, FAC^0 functions are AC^0 -continuous.

2.1 Noninvertibility

We first consider circuits with a pair of inputs $\langle u, v \rangle$, where the represented function is valued v on u if the circuit accepts the pair $\langle u, v \rangle$. By making no syntactic distinction between input and output, any function has the same complexity as its inverse if it is functional. We show that this blurs definability:

Proposition 1. *There is an AC^0 -continuous transduction in FAC^0 whose inverse is functional and not AC^0 -continuous.*

Proof. Consider the minimal, two-state automaton for $L = 0^*(a0^*b0^*)^*$ and turn it into a transducer by letting $\nu(\cdot, 0) = 0$ and $\nu(\cdot, a) = \nu(\cdot, b) = 1$, and call τ the resulting transduction. The FAC^0 circuit for τ first checks that the input is in L . This can be done as $L \in \text{AC}^0$, a fact that can be seen relying on the logical characterization of AC^0 : a word is in L iff its first non-0 letter is an a , its last a b , and the closest non-0 letters to an a (resp. a b) are b 's (resp. a 's). Next, the circuit simply maps 0 to 0 and a, b to 1. The transduction being in FAC^0 , it is AC^0 -continuous. Now let $\sigma = \tau^{-1}$, it is clearly functional. But $\sigma^{-1}(L)$ is PARITY, hence σ is not AC^0 -continuous. \square

Thus, much in the fashion of FAC^0 , this implies that there should be distinguished input and output gates. We next deal with how their lengths are specified.

2.2 Output Length as a Parameter

Aiming for a natural and succinct model, we may want that the family of circuits be parametrized solely by the input length. In such a framework, the presented circuit for a given input length is equipped with a way to “deactivate” output gates, in order to allow for different output lengths. Formalizing this idea further, a *deactivating circuit* C with n inputs and m outputs is an usual circuit with an extra input valued \mathbf{z} , a new constant symbol. This new symbol behaves as follows: $1 \vee \mathbf{z} = \mathbf{z} \vee 1 = 1$, and any other combination of \mathbf{z} with $0, 1, \vee, \wedge, \neg$ is valued \mathbf{z} . The output of C on a given input is its usual output stripped of the \mathbf{z} symbol. The frameworks used in [5, 10, 14] are logic counterparts of this model. Then:

Proposition 2. *There is a transduction expressible as a constant-depth, polysize family of deactivating circuits which is not AC^0 -continuous.*

Proof. The erasing morphism $0 \mapsto \varepsilon, 1 \mapsto 1$ is a transduction τ that can be expressed as a family of circuits as in the statement of the Proposition, but $\tau^{-1}(1^{2\mathbb{N}})$ is PARITY $\notin \text{AC}^0$. \square

We thus reach the following definition, that will serve as a basis for our study:

Definition 1 (Functional Circuits). *A function $\tau: \Sigma^* \rightarrow T^*$ is expressed as a circuit family $(C_m^n)_{n,m \geq 0}$, where C_m^n is a circuit with n inputs and $m + 1$ outputs, if:*

$$(\forall u, v \in \Sigma^*) \quad \tau(u) = v \Leftrightarrow C_m^n|_v^u(u) = (v, 1) .$$

The size of the family is the mapping from \mathbb{N} to $\mathbb{N} \cup \{\infty\}$, defined by $n \mapsto \sup_{m \geq 0} |C_m^n|$. Similarly, the depth of the family is the mapping that associates n to the supremum of the depths of each C_m^n . The class FAC_v^0 , standing for functions in AC^0 with variable output length, is the class of functions expressible as a family of constant-depth, polysize circuits. The class $\text{FAC}_v^0(\mathcal{V})$ is defined in the same fashion as $\text{AC}^0(\mathcal{V})$, and we let $\text{FACC}_v^0 = \text{FAC}_v^0(\text{MOD})$.

Remark 1.

- Any function τ in $\text{FAC}_v^0(\mathcal{V})$ is such that $n \mapsto \max_{u \in \Sigma^n} |\tau(u)|$ has value in \mathbb{N} , that is, for a given input size, there is a finite number of possible output sizes. More precisely, this mapping is polynomially bounded. We show this implies that τ is $\text{AC}^0(\mathcal{V})$ -continuous. Let $(C_m^n)_{n,m \geq 0}$ be the circuit family for τ . Given a language L in $\text{AC}^0(\mathcal{V})$ expressed by the circuit family $(D_n)_{n > 0}$, $\tau^{-1}(L) \cap \Sigma^n$ is recognized by the circuit that applies a polynomial number of circuits C_m^n to the input, and checks that the only m such that C_m^n outputs $(v, 1)$ is such that $v \in D_m$.
- If for any n there is an m such that $\tau(\Sigma^n) \subseteq \Sigma^m$, i.e., if τ is not of variable output length, then $\tau \in \text{FAC}_v^0(\mathcal{V})$ is equivalent to $\tau \in \text{FAC}^0(\mathcal{V})$.
- We will be interested in functions from Σ^* to \mathbb{N} , and will speak of their circuit definability. In this context, the function is either seen as taking value in $\{1\}^*$, and dealt with using a variable-output-length circuit, or taking value in $\{0, 1\}^*$ using an FAC^0 -like circuit, the output value then corresponding to the position of the last 1 in the output. These two views are equivalent, and hence we do not rely on a specific one. We note that (general) transductions from Σ^* to $\{1\}^*$ have been extensively studied in [7]; therein, Choffrut and Schützenberger show that such a function is a transduction iff it has a strong form of *uniform* continuity, akin to the one presented in the introduction, with longest common *subwords* instead of prefixes.

3 Separability, Definability, and Lm-Varieties of Stamps

Recall that the transition monoid of an automaton A is the monoid under composition consisting of the functions $f_w: Q \rightarrow Q$ defined by $f_w(q) = \delta(q, w)$. Historically, regular languages were studied through properties of the transition monoids of their minimal automata (the so-called *syntactic monoids*). As previously mentioned, the minimal automata for $\text{EVEN} \in \text{AC}^0$ and $\text{PARITY} \notin \text{AC}^0$ have the same transition monoid, hence the class $\text{AC}^0 \cap \text{REG}$ admits no syntactic monoid characterization. Starting with [2], the interest shifted to *transition morphisms* of automata, i.e., the surjective morphisms $\varphi: w \mapsto f_w$. It is indeed shown therein that a regular language is in AC^0 iff $\varphi(\Sigma^s) \cup \{\varphi(\varepsilon)\}$ is an aperiodic monoid for some $s > 0$ and φ associated with the minimal automaton.

A *stamp* is a surjective morphism from a free monoid to a finite monoid. A systematic study of the classes of languages described by stamps turned out to be a particularly fruitful research endeavor of the past decade [6, 9, 15, 18]. Our use of this theory will however be kept minimal, and we will strive to only appeal to it in this section. The goal of the forthcoming Lemma 1 is indeed to express algebraic properties in a language-theoretic framework only.

Given a stamp $\varphi: \Sigma^* \rightarrow M$, we say that L is *recognized* by φ if there is a set $E \subseteq M$ such that $L = \varphi^{-1}(E)$ —in this case, we also say that L is recognized by M , which corresponds to the usual definition of recognition (e.g., [17]). We say that a stamp $\varphi: \Sigma^* \rightarrow M$ *lm-divides* a stamp $\psi: T^* \rightarrow N$ if $\varphi = \eta \circ \psi \circ h$, where $h: \Sigma^* \rightarrow T^*$ is an lm-morphism and $\eta: N \rightarrow M$ is a partial surjective

morphism. The *product* of two stamps φ and ψ with the same domain Σ^* is the stamp mapping $a \in \Sigma$ to $(\varphi(a), \psi(a))$. Finally, an lm-variety of stamps is a class of stamps containing the stamps $\Sigma^* \rightarrow \{1\}$ and closed under lm-division and product. An Eilenberg theorem holds for lm-varieties: there is a one-to-one correspondence between lm-varieties of stamps and the lm-varieties of languages they recognize [18]. We show:

Lemma 1. *Let A be an automaton, \mathbf{V} an lm-variety of stamps, and \mathcal{V} its corresponding lm-variety of languages. The following are equivalent:*

- (i) *The transition morphism of A is in \mathbf{V} ;*
- (ii) *A is \mathcal{V} -all-definable;*
- (iii) *A is \mathcal{V} -all-separable.*

Proof. (i) \rightarrow (ii). Let φ be the transition morphism of A . Then $L(A, q) = \varphi^{-1}(E)$ where $E = \{f_w \mid f_w(q_0) = q\}$, hence $L(A, q) \in \mathcal{V}$.

(ii) \rightarrow (iii). This is immediate, as $L(A, q)$ separates q from any other state.

(iii) \rightarrow (i). Write $L_{q,q'}$ for the language separating q from q' . As each of these are recognized by stamps in \mathbf{V} and \mathbf{V} is closed under product, the language $L_q = \bigcap_{q' \neq q} L_{q,q'}$ is also recognized by a stamp in \mathbf{V} . Similarly, taking the product of the stamps recognizing the different L_q 's, we see that all of the L_q 's are recognized by the same stamp $\psi: \Sigma^* \rightarrow N$ in \mathbf{V} ; let thus E_q be such that $L_q = \psi^{-1}(E_q)$. Let $\varphi: \Sigma^* \rightarrow M$ be the transition morphism of A . We claim that φ lm-divides ψ , concluding the proof as \mathbf{V} is closed under lm-division. Define $\eta: N \rightarrow M$ by $\eta(\psi(w)) = \varphi(w)$. If η is well-defined, then it is a surjective morphism, and we are done as $\varphi = \eta \circ \psi$. Suppose $\varphi(u) \neq \varphi(v)$, then there is a $p \in Q$ such that $\delta(p, u) = q$ and $\delta(p, v)$ is either undefined or a state $q' \neq q$. Let w be a word such that $\delta(q_0, w) = p$, then $\psi(wu) \in E_q$ and $\psi(wv) \notin E_q$, hence $\psi(u) \neq \psi(v)$, showing that η is well-defined. \square

Remark 2. For $\mathcal{V} = \mathcal{QA}$ and $\mathcal{V} = \mathcal{M}_{\text{sol}}$, the properties of Lemma 1 are decidable.

As advertised, the rest of this paper will now be free from (lm-varieties of) stamps except for a brief incursion when discussing our results in Sect. 6. Lemma 1 enables a study that stands in the algebraic tradition with no appeal to its tools.

4 The Transductions in $\text{FAC}_{\mathcal{V}}^0(\mathcal{V})$

In sharp contrast with the work of Reutenauer and Schützenberger [16], we are especially interested in the shape of the outputs of the transduction. It turns out that most of its complexity is given by the following output-length function:

Definition 2 ($\tau_{\#}$). *Let τ be a transduction. The function $\tau_{\#}: \Sigma^* \rightarrow \mathbb{N}$ is the output-length function of $\text{MinT}(\tau)$ with all the states deemed final. In symbols, $\tau_{\#}(w) = |\nu(q_0, w)|$, with $\text{MinT}(\tau)$ as the underlying transducer.*

Theorem 1. *Let τ be a transduction and \mathcal{V} be such that $\text{AC}^0(\mathcal{V}) \cap \text{REG} = \mathcal{V}$. The following constitutes a chain of implications:*

- (i) $\tau \in \text{FAC}_v^0(\mathcal{V})$;
- (ii) τ is $\text{AC}^0(\mathcal{V})$ -continuous;
- (iii) τ is \mathcal{V} -continuous;
- (iv) $\text{MinT}(\tau)$ is \mathcal{V} -all-definable.

Moreover, if $\tau_{\#} \in \text{FAC}_v^0(\mathcal{V})$ then (iv) implies (i). Somewhat conversely, (i) implies $\tau_{\#} \in \text{FAC}_v^0(\mathcal{V})$.

Proof. (i) \rightarrow (ii). This was alluded to in Remark 1.

(ii) \rightarrow (iii). This follows from the closure under inverse transductions of REG and the hypothesis that the regular languages of $\text{AC}^0(\mathcal{V})$ are in \mathcal{V} .

(iii) \rightarrow (iv). Let q, q' be two states of $A = \text{MinT}(\tau)$. We show that we can separate q from q' . We distinguish the following cases, that span all the possibilities thanks to the output-minimality of A . In each case, we build a language L separating $L(A, q)$ and $L(A, q')$, with $L \in \mathcal{V}$ relying on continuity and on \mathcal{V} being an lmv-variety by hypothesis. By Lemma 1, we then conclude (iv).

- *Case 1:* There is a w such that only one of $\delta(q, w)$ or $\delta(q', w)$ is in F . We suppose $\delta(q, w) \in F$, without loss of generality as \mathcal{V} is closed under complement. Let $L = (\tau^{-1}(T^*)w^{-1})$, a word x is in L iff $\delta(q_0, xw) \in F$. This is the case for all words in $L(A, q)$ and for none in $L(A, q')$, hence L separates these languages.
- *Case 2:* There is a w such that both $\delta(q, w)$ and $\delta(q', w)$ are in F , and words u, u' such that $\nu(q, w) = u \neq u' = \nu(q', w)$. Then we have two possibilities:
 - *Case 2.1:* If $|u| = |u'|$. For a word $x \in \Sigma^*$, if $\tau(xw)$ ends with u , then $\delta(q_0, x)$ cannot be q' . Hence $L = (\tau^{-1}(T^*u))w^{-1} \in \mathcal{V}$ separates $L(A, q)$ from $L(A, q')$.
 - *Case 2.2:* If $|u| \neq |u'|$. Define $k \in \mathbb{N}$ to be such that $|u| \not\equiv |u'| \pmod{k}$, and let $s \in \{0, 1, \dots, k-1\}$. Let $x \in \Sigma^*$ be such that $s \equiv |\nu(q_0, x)| \pmod{k}$. Then if $\delta(q_0, x) = q$, we have $|\tau(xw)| \equiv s + |u| \pmod{k}$, while $\delta(q_0, x) = q'$ makes this equation false. Hence the union L over every s of the languages $(\tau^{-1}(T^{k\mathbb{N}+s+|u|}))w^{-1}$ separates $L(A, q)$ from $L(A, q')$.

(iv) \rightarrow (i), assuming $\tau_{\#} \in \text{FAC}_v^0(\mathcal{V})$. We construct an $\text{FAC}_v^0(\mathcal{V})$ circuit family for τ . Fix an input size n and an output size m . Given an input $x = x_1x_2 \cdots x_n$, we first check, using $\tau_{\#}$, that the output length of τ on x is indeed m , and wire this answer properly to the $(m+1)$ -th output bit. Next, the j -th output bit, $1 \leq j \leq m$, is computed as follows. We apply $\tau_{\#}$ to every prefix of x , until we find an i such that $\tau_{\#}(x_{<i}) < j \leq \tau_{\#}(x_{\leq i})$, where $x_{<i} = x_1x_2 \cdots x_{i-1}$ and similarly for $x_{\leq i}$. Relying on the languages $L(A, q)$, we find the state q in $\text{MinT}(\tau)$ reached by $x_{<i}$, and let $u = \nu(q, x_i)$. The j -th output bit then corresponds to the $(j - \tau_{\#}(x_{<i}))$ -th letter of u .

(i) \rightarrow ($\tau_{\#} \in \text{FAC}_v^0(\mathcal{V})$). Suppose (i), this implies (iv). We construct an $\text{FAC}_v^0(\mathcal{V})$ circuit family for $\tau_{\#}$. Fix the input size n , and let $x = x_1x_2 \cdots x_n$ be the input. We can check, using the languages $L(\text{MinT}(\tau), q)$, in which state q the word x ends when read. Let w_q be a fixed word such that $\delta(q, w_q) \in F$, and let $r = |\nu(q, w_q)|$. It suffices now to plug the word xw_q in the circuit for τ ; the value of $\tau_{\#}(x)$ is then the length of $\tau(xw_q)$ minus r . \square

Remark 3. The proof of Theorem 1 shows that $\text{MinT}(\tau)$, and hence $\tau_{\#}$, can be arbitrarily chosen as long as it is output-minimal. The role of $\tau_{\#}$ is discussed at greater length in Sect. 6.

5 An Application to AC^0 and ACC^0

Our primary focus is on the *decidability* of the membership of transductions in small-complexity classes. Theorem 1, while providing a characterization of these transductions, does not come with a decidable property in the general case—even when some conjectured separations are presupposed. With AC^0 and ACC^0 , the functions $\tau_{\#}$ that can be expressed with circuits can however be characterized.

Definition 3 (Constant Ratio). *A transducer has constant ratio if every two words of the same length looping on a state produce outputs of the same length from this state. In symbols, for any state q and any words u, v of the same length, $\delta(q, u) = \delta(q, v) = q$ implies $|\nu(q, u)| = |\nu(q, v)|$.*

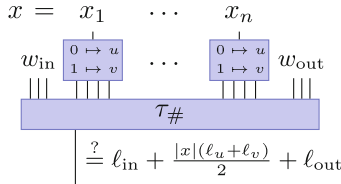
Remark 4. The name of the latter property stems from the fact that in such a transducer, for any state q , there is a ratio θ such that if $\delta(q, u) = q$, then $|\nu(q, u)| = \theta|u|$. Indeed, suppose a transducer has constant ratio, and let u and v be words with $\delta(q, u) = \delta(q, v) = q$ for some q . Write $|\nu(q, u)| = \theta_1|u|$ and $|\nu(q, v)| = \theta_2|v|$. Then $x = u^{|v|}$ and $y = v^{|u|}$ are of the same length, and $\theta_1|u| \times |v| = |\nu(q, x)| = |\nu(q, y)| = \theta_2|v| \times |u|$, hence $\theta_1 = \theta_2$.

Lemma 2. (Assuming $\text{ACC}^0 \neq \text{TC}^0$.) *Let τ be a transduction. If $\tau_{\#}$ is in FACC_v^0 , then $\text{MinT}(\tau)$ has constant ratio.*

Proof. Suppose that $A = \text{MinT}(\tau)$ does not have constant ratio. We give a circuit family in AC^0 with $\tau_{\#}$ -gates for the language $L = \{w \in \{0, 1\}^* \mid |w|_0 = |w|_1\}$, which is complete for TC^0 . Hence $\tau_{\#}$ cannot admit an FACC_v^0 circuit family.

As A does not have constant ratio, there are a state q in A and two words $u, v \in \Sigma^*$ of the same length, such that $\delta(q, u) = \delta(q, v) = q$ and $\ell_u = |\nu(q, u)|$ is different from $\ell_v = |\nu(q, v)|$. Further, let w_{in} (resp. w_{out}) be such that $\delta(q_0, w_{\text{in}}) = q$ (resp. $\delta(q, w_{\text{out}}) \in F$), and let $\ell_{\text{in}} = |\nu(q_0, w_{\text{in}})|$ (resp. $\ell_{\text{out}} = |\nu(q, w_{\text{out}})|$).

We describe the circuit for L for input size n . Let x denote the input. First, the circuit transforms each 0 into u , and each 1 into input. The circuit can be graphically represented as follows:



First, the circuit transforms each 0 into u , and each 1 into v —this can be done as $|u| = |v|$. Then w_{in} is prepended and w_{out} appended to it, and the resulting word x' is fed to $\tau_{\#}$. The output is $\ell_{\text{in}} + |x|_0 \times \ell_u + |x|_1 \times \ell_v + \ell_{\text{out}}$, that is:

$$\tau_{\#}(x') = \ell_{\text{in}} + \frac{1}{2}(|x|(\ell_u + \ell_v) + (|x|_0 - |x|_1)(\ell_u - \ell_v)) + \ell_{\text{out}} .$$

Now $(|x|_0 - |x|_1)(\ell_u - \ell_v)$ cancels out iff x has as many 0's as 1's. Hence $x \in L$ iff the output of $\tau_{\#}$ is $\ell_{\text{in}} + \frac{1}{2}|x|(\ell_u + \ell_v) + \ell_{\text{out}}$, which is verifiable in AC^0 . \square

Having a constant ratio provides an easy way to compute the length function:

Lemma 3. *Let τ be a transduction. If $\text{MinT}(\tau)$ has constant ratio and is \mathcal{V} -all-definable, then $\tau_{\#}$ is in $\text{FAC}_v^0(\mathcal{V})$.*

Proof. The circuit for $\tau_{\#}$ guesses a path without cycles for the input word, and checks that, modulo cycling, it is indeed a correct path for it. The output value is then entirely determined by the positions of the input at which this underlying path is taken. We now give a more precise construction.

Let $\pi = (q_0, a_0)(q_1, a_1) \cdots (q_k, a_k) \in (Q \times \Sigma)^*$ be an *accepting simple path* in $A = \text{MinT}(\tau)$, that is, $q_{i+1} = \delta(q_i, a_i)$, $q_{k+1} = \delta(q_k, a_k) \in F$, and for $i \neq j$, $q_i \neq q_j$. There is a finite number of such paths, and for each of them, the circuit contains the following subcircuit. The subcircuit checks that the path in A for the input word follows π , that is, if all the cycles are removed, then the resulting path is π . To do so, for each possible values of $1 \leq p_0 < p_1 < \cdots < p_k \leq n$ such that $\sum p_i = n$ (there is a polynomial number of them), the subcircuit checks for all $i \leq k$ that the prefix of length $p_i - 1$ of the input is in $L(A, q_i)$, and that the input at position p_i is a_i . If this holds for all i , then the input word follows the path π , possibly cycling on each of the states q_i , $i \leq k + 1$, and the output length is entirely determined. Indeed, with $\theta_0, \theta_1, \dots, \theta_{k+1}$ the ratios of q_0, q_1, \dots, q_{k+1} respectively, and ℓ the sum of the output lengths of the transitions in π (i.e., $\tau_{\#}(a_0 a_1 \cdots a_k)$), the value of $\tau_{\#}$ on the input is:

$$\theta_0 \times (p_0 - 1) + \sum_{i=1}^k \theta_i \times (p_i - p_{i-1} - 1) + \theta_{k+1} \times (n - p_k) + \ell. \quad \square$$

Corollary 1. *Let τ be a transduction. The following are equivalent, where the “resp.” part assumes $\text{ACC}^0 \neq \text{TC}^0$:*

- (i) $\tau \in \text{FAC}_v^0$ (resp. $\in \text{FACC}_v^0$);
- (ii) τ is continuous for AC^0 (resp. for ACC^0) and $\text{MinT}(\tau)$ has constant ratio;
- (iii) τ is continuous for \mathcal{QA} (resp. for \mathcal{M}_{sol}) and $\text{MinT}(\tau)$ has constant ratio;
- (iv) $\text{MinT}(\tau)$ is all-definable for \mathcal{QA} (resp. for \mathcal{M}_{sol}) and has constant ratio.

Remark 5. It should be noted that the choice of $\text{MinT}(\tau)$ is again irrelevant. Either all the output-minimal transducers for τ are constant ratio, or none are.

Theorem 2. *It is decidable whether a transducer realizes an FAC_v^0 function. If it does, then a circuit family can be constructed. The same holds for FACC_v^0 assuming $\text{ACC}^0 \neq \text{TC}^0$.*

Proof. This is a direct consequence of Corollary 1, together with the minimization algorithm of [8], the fact that \mathcal{V} -all-definability is decidable, and the fact that it can be checked that a transducer has constant ratio: it is indeed enough to check the property on cycles that do not go twice in the same state except for the first. The constructions of Theorem 1 and Lemma 3 are then effective. \square

Corollary 1 can be slightly strengthened for AC^0 , as in this case:

Proposition 3. *If τ is \mathcal{QA} -continuous, then $\text{MinT}(\tau)$ has constant ratio.*

Proof. This is a variant of Lemma 2, where we only rely on the inverse image of τ instead of a full circuit construction.

Suppose that $A = \text{MinT}(\tau)$ does not have constant ratio. There are a state q in $A = \text{MinT}(\tau)$ and two words $u, v \in \Sigma^*$ of the same length, such that reading u (resp. v) from q produces an output of length ℓ_u (resp. ℓ_v), and $\ell_u < \ell_v$. Now the words $y = u^{\ell_v}$ and $z = v^{2\ell_u}$ are such that y produces an output of size $\ell_y = \ell_u \times \ell_v$, and z produces an output of size $\ell_z = \ell_v \times 2\ell_u = 2\ell_y$.

Now if τ is a \mathcal{QA} -continuous transduction, so is the function τ' mapping $x \in \{0, 1\}^*$ to a word on $\{a\}^*$ with $\ell_y \times |x|_1 + \ell_z \times |x|_0$ letters a —it is simply a matter of replacing 1 with y , 0 with z , and correctly reaching the state q . But $\tau'^{-1}(a^{2\ell_y \mathbb{N}})$ is PARITY: indeed, x has an odd number of 1 iff $\tau'(x)$ contains an odd number of blocks a^{ℓ_y} . Hence τ' is not \mathcal{QA} -continuous, and neither is τ . \square

6 Discussion and Limitations

1 We note that Proposition 3 fails in the case of ACC^0 , as the following example shows. Consider the morphism $h: a \mapsto a, b \mapsto aa$. As the regular languages of ACC^0 , \mathcal{M}_{sol} , are closed under inverse morphism (this is a consequence of \mathcal{M}_{sol} being a variety), h is \mathcal{M}_{sol} -continuous. However, $\text{MinT}(h)$ does not have constant ratio. This was already noted in a different setting by Lange and McKenzie [13].

2 The major role that $\tau_{\#}$ plays in Theorem 1 raises several questions. First, is it the case that all the complexity of a transduction is characterized by its length function? In symbols, is it true that $\tau_{\#} \in \text{FAC}_{\mathcal{V}}^0(\mathcal{V}) \Rightarrow \tau \in \text{FAC}_{\mathcal{V}}^0(\mathcal{V})$? The following example shows that it is not. Consider the transduction from $\{0, 1\}^*$ to $\{a, b\}^*$ that outputs a if the word read so far is in PARITY, and b otherwise. Then $\tau_{\#}$ is total and maps every word to its length, it is thus in $\text{FAC}_{\mathcal{V}}^0$. However, w is in PARITY iff the last letter of $\tau(w)$ is an a , that is, $\tau^{-1}(\{a, b\}^*a)$ is PARITY, hence τ is not \mathcal{QA} -continuous, thus cannot be in $\text{FAC}_{\mathcal{V}}^0$.

Next, going down two levels in the statement of Theorem 1, we may wonder whether the \mathcal{V} -continuity of τ is equivalent to that of $\tau_{\#}$. One direction is true, but its converse fails, as the previous example shows:

Proposition 4. *If τ is \mathcal{V} -continuous, then so is $\tau_{\#}$.*

Proof. Suppose τ is \mathcal{V} -continuous. Let E be a set of integers, and write Σ^E for the words of lengths in E . Suppose $\{a\}^E$ is in \mathcal{V} ; we show $\tau_{\#}^{-1}(E) \in \mathcal{V}$.

From Theorem 1, $A = \text{MinT}(\tau)$ is \mathcal{V} -all-definable. Let q be a state of A , and w a word mapping q to a final state while outputting u . Then $(\tau^{-1}(\Sigma^E.u))w^{-1} \cap L(A, q)$ is in \mathcal{V} , as τ is \mathcal{V} -continuous and $\{a\}^E \in \mathcal{V}$. Now the union of all these sets for all states q is precisely $\tau_{\#}^{-1}(E)$, hence it is in \mathcal{V} . \square

3 Our interest in circuits obscured an equally interesting problem: characterizing the \mathcal{V} -continuous transductions. A general question raised by our characterization is:

Question 1. Which lm-varieties \mathcal{V} verify the following statement? A transduction τ is \mathcal{V} -continuous iff $\text{MinT}(\tau)$ is \mathcal{V} -all-definable and $\tau_{\#}$ is \mathcal{V} -continuous.

A direct consequence of Proposition 3 is that $\mathcal{V} = \mathcal{QA}$ verifies Question 1. Another such class is given in [16]; therein, Reutenauer and Schützenberger show that the property holds for $\mathcal{V} = \mathcal{G}$, the (lm-)variety of group languages, that is, languages with a group as syntactic monoid. More precisely, they show that τ is \mathcal{G} -continuous iff the transition monoid of $\text{MinT}(\tau)$ is a group; this latter property is equivalent to: the transition morphism of $\text{MinT}(\tau)$ is a stamp $\Sigma^* \rightarrow G$, for G a group. The set of such stamps is an lm-variety of stamps (see [6]), thus by Lemma 1, their characterization is indeed of the form of Question 1.

4 It is interesting to note that the property on $\tau_{\#}$ of Question 1 vanishes for groups: this can be seen as a consequence of Reutenauer and Schützenberger’s characterization itself, as $\tau_{\#}$ has the same transition monoid as $\text{MinT}(\tau)$. On the other hand it is shown in the same article that there are transductions with an aperiodic monoid that are not continuous for aperiodic languages. This raises the question:

Question 2. Which lm-varieties \mathcal{V} verify the following statement? If a transduction τ is such that $\text{MinT}(\tau)$ is \mathcal{V} -all-definable, then $\tau_{\#}$ is \mathcal{V} -continuous.

5 Recall that a nondeterministic transduction is functional iff it is realized by an unambiguous transduction (see, e.g., [3]). As circuits can read the input multiple times and in any direction, it seems that they can handle deterministic and unambiguous transductions in the same fashion. Hence a generalization of Theorem 1 to the unrestricted case of functional transductions should hold.

Acknowledgment. We thank Michael Blondin, Michael Hahn, and the referees.

References

1. Barrington, D.A.M.: Bounded-width polynomial size branching programs recognize exactly those languages in NC^1 . *J. Comp. Syst. Sc.* **38**, 150–164 (1989)
2. Barrington, D.A.M., Compton, K., Straubing, H., Thérien, D.: Regular languages in NC^1 . *J. Comput. Syst. Sci.* **44**(3), 478–499 (1992)
3. Berstel, J.: *Transductions and Context-Free Languages*, Leitfäden der Angewandten Mathematik und Mechanik LAMM. Teubner, Stuttgart (1979)
4. Beyersdorff, O., Datta, S., Krebs, A., Mahajan, M., Scharfenberger-Fabian, G., Sreenivasaiah, K., Thomas, M., Vollmer, H.: Verifying proofs in constant depth. *TOCT* **5**(1), 2 (2013)
5. Bojańczyk, M.: Transducers with Origin Information. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) *ICALP 2014, Part II*. LNCS, vol. 8573, pp. 26–37. Springer, Heidelberg (2014)
6. Chaubard, L., Pin, J.É., Straubing, H.: First-order formulas with modular predicates. In: *LICS*, pp. 211–220. IEEE (2006)

7. Choffrut, C., Schützenberger, M.P.: Counting with rational functions. *Theor. Comput. Sci.* **58**(1–3), 81–101 (1988)
8. Choffrut, C.: A generalization of Ginsburg and Rose’s characterization of G-S-M mappings. In: Maurer, H.A. (ed.) *ICALP 1979*. LNCS, vol. 71, pp. 88–103. Springer, Heidelberg (1979)
9. Esik, Z., Ito, M.: Temporal logic with cyclic counting and the degree of aperiodicity of finite automata. *Acta Cybern.* **16**(1), 1–28 (2003)
10. Filiot, E., Krishna, S.N., Trivedi, A.: First-order definable string transformations. In: Raman, V., Suresh, S.P. (eds.) *FSTTCS. LIPIcs*, vol. 29, pp. 147–159 (2014)
11. Furst, M., Saxe, J.B., Sipser, M.: Parity, circuits, and the polynomial-time hierarchy. *Theor. Comput. Syst.* **17**, 13–27 (1984)
12. Koucký, M., Pudlák, P., Thérien, D.: Bounded-depth circuits: separating wires from gates. In: *STOC*, pp. 257–265. ACM (2005)
13. Lange, K.-J., McKenzie, P.: On the complexity of free monoid morphisms. In: Chwa, K.-Y., Ibarra, O.H. (eds.) *ISAAC 1998*. LNCS, vol. 1533, pp. 247–255. Springer, Heidelberg (1998)
14. Lautemann, C., McKenzie, P., Schwentick, T., Vollmer, H.: The descriptive complexity approach to LOGCFL. *J. Comput. Syst. Sci.* **62**(4), 629–652 (2001)
15. Pin, J.É., Straubing, H.: Some results on C-varieties. *RAIRO-Theor. Inf. Appl.* **39**(01), 239–262 (2005)
16. Reutenauer, C., Schützenberger, M.P.: Variétés et fonctions rationnelles. *Theor. Comput. Sci.* **145**(1–2), 229–240 (1995)
17. Straubing, H.: *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Boston (1994)
18. Straubing, H.: On logical descriptions of regular languages. In: Rajsbaum, S. (ed.) *LATIN 2002*. LNCS, vol. 2286, pp. 528–538. Springer, Heidelberg (2002)
19. Vollmer, H.: *Introduction to Circuit Complexity*. Springer-Verlag, Berlin (1999)

Locally Chain-Parsable Languages

Stefano Crespi Reghizzi¹, Violetta Lonati²,
Dino Mandrioli¹, and Matteo Pradella¹(✉)

¹ DEIB - Politecnico di Milano, Milan, Italy
{stefano.crespireghizzi,dino.mandrioli,matteo.pradella}@polimi.it

² DI - Università degli Studi di Milano, Milan, Italy
lonati@di.unimi.it

Abstract. If a context-free language enjoys the local parsability property then, no matter how the source string is segmented, each segment can be parsed independently, and an efficient parallel parsing algorithm becomes possible. The new class of locally chain-parsable languages (LCPL), included in deterministic context-free languages, is here defined by means of the chain-driven automaton and characterized by decidable properties of grammar derivations. Such automaton decides to reduce or not a factor in a way purely driven by the terminal characters, thus extending the well-known concept of Input-Driven (ID) (visibly) pushdown machines. LCPL extend and improve the practically relevant operator-precedence languages (Floyd), which are known to strictly include the ID languages, and for which a parallel-parser generator exists. Consistently with the classical results for ID, chain-compatible LCPL are closed under reversal and Boolean operations, and language inclusion is decidable.

1 Introduction

Syntax analysis or parsing of context-free (CF) languages is a mature research area, and good parsing algorithms are available for the whole CF family and for the deterministic subfamily (DCFL) that is of concern here. Yet the classical parsers are strictly serial and cannot profit from the parallelism of current computers. An exception is the parallel deterministic parser [2, 3] based on Floyd's [7] operator-precedence grammars (OPG) and their languages (OPL), which are included in DCFL. This is a data-parallel algorithm that is based on a theoretical property of OPG, called *local parsability*: any arbitrary factor of a sentence can be deterministically parsed, returning the unique partial syntax-tree whose frontier is the input string.

LL(k) and LR(k) grammars do not have this property, and their parsers must scan the input left-to-right to build leftmost derivations (or reversed-rightmost ones). On the contrary, the abstract recognizer of a locally parsable language, called a *local parser*, repeatedly looks in some arbitrary position inside the input

Partially supported by PRIN 2010LYA9RH-006, and CNR-IEIIT.

string for a production right-hand side (RHS) and reduces it. The local parsability property ensures the correctness of the syntax tree thus obtained, no matter the position of reduction applications.

The informal idea of local parsability is occasionally mentioned in old research on parallel parsing, and has been formalized for OPG in [2]. Our first contribution is to propose the definition of a new and more general class of locally parsable languages: the language family to be called *Locally Chain-Parsable* (LCPL), which gains in generative capacity and bypasses some inconveniences of OPG.

The other contribution is towards a generalization of the well-known family of input-driven (alias visibly push-down) languages (IDL) [1, 11], which are characterized by push-down machines that choose to perform a push/pop/stay operation depending on the alphabetic class (opening/closing/internal) of the current input character, without a need to check the top of stack symbol. Since the attention IDL have recently attracted is due to their rich closure and decidability properties, we hope that the introduction of a larger family of languages with similar properties may be also of interest.

To understand in what sense our LCPL are input-driven, we first recall that IDL generalize parenthesis languages, by taking the opening/closing characters as parentheses to be balanced, while the internal characters are handled by a finite-state automaton. It suffices a little thought to see that IDL have the local parsability property, a fact also stemming from the fact that IDL are included in OPL [6]. Yet, the rigid alphabetic 3-partition severely reduces their generative capacity. If we allow the parser decision whether to push, pop, or stay, to be based on a *pair* of adjacent terminal characters (more precisely on the precedence relation \langle, \rangle, \doteq between them), instead of just one as in the IDL, we obtain the OPL family, which has essentially the same closure and decidability properties [6, 9]. Loosely speaking, we may say that the input that drives the automaton for OPL is a terminal string of length two.

With the LCPL definition, we move further: the automaton bases its decision whether to reduce or not a factor (which may contain nonterminals) on the purely terminal string orderly containing: the preceding terminal, the terminals of the factor, and the following terminal. Such triplet will be called a *chain* and the machine a chain-driven automaton. For a given CF grammar, the length of chains has an upper bound, which bounds the input portion that drives the choice of a move by the recognizer.

The paper is organized as follows. After the Preliminaries, Sect. 3 introduces the chain-driven machine as a recognizer for CF. Sect. 4 defines local chain parsability for chain-driven automata and for grammars, and proves the two notions to be equivalent; Sect. 4.1 extends the definition of chains and formulates a decidability condition for local chain parsability based on the absence of conflicts; Sect. 4.2 proves, among others, the Boolean closure and decidability properties of LCPL. Section 5 establishes the strict inclusion of OPL (and hence also IDL) within LCPL, and claims through a practical example that LCPG are more suitable than OPG for specifying real programming languages. Section 6 is on related work and draw some conclusions.

2 Preliminaries

For terms not defined here, we refer to any textbook on formal languages, e.g. [8]. The *terminal* alphabet is denoted by Σ ; it includes the letter $\#$ used as start and end of text. Let Δ be an alphabet disjoint from Σ . A string $\beta \in (\Sigma \cup \Delta)^*$ is in *operator form* if it contains one or more terminals and does not contain a factor from Δ^2 , i.e., no adjacent symbols from Δ ; $\text{OF}(\Delta)$ denotes the set of all operator form strings over $\Sigma \cup \Delta$.

A *context-free* grammar is a 4-tuple $G = (V_N, \Sigma, P, S)$, where V_N is the nonterminal alphabet, P the set of rules, and $S \subseteq V_N$ is the set of axioms. The *total* alphabet is $V = V_N \cup \Sigma$. The *stencil* of a rule $A \rightarrow \alpha$ is the rule $N \rightarrow \sigma(\alpha)$, where $\sigma : V_N \rightarrow \{N\}$ maps every nonterminal to the new symbol $N \notin V$.

The *derivation relation* for a grammar G is denoted as usual by \Rightarrow_G and its reflexive and transitive closure by $\xrightarrow{*}_G$. The set of *sentential forms* (s.f.) generated by G is $\text{SF}_G = \{\alpha \in V^* \mid T \xrightarrow{*}_G \alpha, T \in S\}$ and the language generated is $L(G) = \text{SF}_G \cap \Sigma^*$. A grammar is *invertible* if no two rules have identical r.h.s. A grammar is an *operator grammar* (OG) if all r.h.s.'s are in $\text{OF}(V_N)$. Any CF grammar that does not generate ε admits an equivalent OG, which can be assumed to be invertible [8]. Clearly, every s.f. of an OP grammar is in $\text{OF}(V_N)$. In this paper we deal only with reduced OG.

The following naming convention is adopted, unless otherwise specified: lowercase Latin letters a, b, \dots denote terminal characters; uppercase Latin letters A, B, \dots denote nonterminal characters; lowercase Latin letters x, y, z, \dots denote terminal strings; and Greek lowercase letters α, \dots, ω denote strings over $\Sigma \cup V_N$.

We use bold symbols to denote strings over an alphabet that includes the square brackets, e.g. $\mathbf{x} \in (\Sigma \cup \{[,]\})^*$, $\mathbf{\alpha} \in (\Sigma \cup V_N \cup \{[,]\})^*$. We introduce the following short notation for frequently used operations based on projections: for erasing all nonterminal symbols in a string α , we write $\widehat{\alpha}$; for erasing all square brackets, we write $\widetilde{\alpha}$; moreover, $\mathbf{\alpha} \doteq \mathbf{\beta}$ stands for $\widehat{\mathbf{\alpha}} = \widehat{\mathbf{\beta}}$ and $\mathbf{\alpha} \cong \mathbf{\beta}$ stands for $\widetilde{\mathbf{\alpha}} = \widetilde{\mathbf{\beta}}$. Also, we use $\mathbf{\alpha}_{\text{first}}$ and $\mathbf{\alpha}_{\text{last}}$ for taking the first or last symbol in Σ from a string $\mathbf{\alpha}$. The same notation is applied when V_N is replaced by the state set of a machine.

For a CF grammar G , the associated *parenthesis grammar*, denoted by $[G]$, is obtained by bracketing with '[' and ']' each r.h.s. of a rule of G . A grammar G is *structurally ambiguous* if there exists $\mathbf{x}_1 \neq \mathbf{x}_2 \in L([G])$ such that $\mathbf{x}_1 \cong \mathbf{x}_2$. Two grammars G, G' are *structurally equivalent* if $L([G]) = L([G'])$.

3 Chain-Driven Automata

In this section, we present the core formalism of this paper, i.e., the chain-driven automaton, that can be seen as an abstract parser for CF languages. As stated in the introduction such type of abstract parser is particularly well-suited to exploit parallel implementation. First we give and illustrate by example the formal definition of chain-driven automaton, then we prove the equivalence between chain-driven automata and CF grammars.

The key driver in the search for a string to be reduced is the concept of chain. According with the general philosophy of input-driven languages and other similar families, such as, e.g., OPL, where the parsing actions by the recognizing automata are determined exclusively on the basis of terminal characters, the chains driving our automata contain only terminal characters

Definition 1. A chain is a triple $a\langle y\rangle b$ with $a, b \in \Sigma$ and $y \in \Sigma^+$; (a, b) is the context and y the body of the chain.

A chain-driven automaton works by reducing the input string through a sequence of reductions driven by a given set of chains; the automaton finds a given chain within the input string and replaces its body with a state; then the mechanism is applied recursively to the obtained string. Hence during the reduction steps the input string is shortened and simultaneously enriched by the computed states; chains being defined over the input alphabet, the portion of the input factor to be reduced is detected depending on input symbols only; enriching states are used then to (nondeterministically) determine which state will replace the detected factor.

Definition 2. A chain-driven automaton is a tuple $(\Sigma, Q, C, \delta, F)$ where

- Σ is the input alphabet;
- Q is a finite set of states;
- C is a finite set of chains;
- $\delta : \Sigma \times \text{OF}(Q) \times \Sigma \rightarrow \mathcal{P}(Q)$ is the reduce function, where $\delta(a, \gamma, b) \neq \emptyset$ implies $a\langle \hat{\gamma} \rangle b \in C$.
- $F \subseteq Q$ is the set of final states.

A configuration of the automaton is a string $\gamma \in \text{OF}(Q)$. The initial configuration on input $x \in \Sigma^*$ is defined as $\#x\#$; a configuration $\#q\#$ with $q \in F$ is called an accepting configuration. The reduction move is defined as follows: the automaton may perform the move

$$\alpha a\gamma b \beta \xrightarrow{a\langle y\rangle b} \alpha aqb \beta$$

where $\hat{\gamma} = y$, and $\delta(a, \gamma, b) \ni q$. Hence, a move of the automaton deletes a factor in $\text{OF}(Q)$ (corresponding to the body of the chain in C , possibly enriched with states in Q) and replaces it with a state.

A computation of the automaton is a sequence $K_0 \xrightarrow{c_1} K_1 \xrightarrow{c_2} K_2 \xrightarrow{c_3} \dots \xrightarrow{c_n} K_n$ where K_i are configurations, c_i are chains. When not relevant we omit the chain and write simply $K_1 \xrightarrow{*} K_2$. We also use $\xrightarrow{*}$ to denote the reflexive and transitive closure of $\xrightarrow{\quad}$. The language accepted by the automaton is defined as $L(\mathcal{A}) = \{x \in \Sigma^* \mid \#x\# \xrightarrow{*} \#q\# \text{ with } q \in F\}$.

Example 1. Consider the language of arithmetic expressions on $\{e, +, *\}$ with the obvious meaning of symbols. A chain-driven automaton recognizing such expressions can be defined as follows: C contains chains $\#(+)\#, \#(+)+, \#(*)\#, \#(*)+, +(*)\#, +(*)+, +(*)*, \#(*)*$, and all chains $a\langle e\rangle b$ with $a, b \in \{\#, *, +\}$;

$Q = \{q_e, q_+, q_*\}$, $F = Q$, and δ is given in the following table, where the first column collects the contexts (a, b) , and the second row specifies the strings in $OF(Q)$ gathered according to their projection.

	$\hat{\gamma} = *$	$\hat{\gamma} = +$							$\hat{\gamma} = e$
	$q_e * q_e$	$q_* * q_e$	$q_e + q_e$	$q_* + q_e$	$q_+ + q_e$	$q_e + q_*$	$q_* + q_*$	$q_+ + q_*$	e
$(\#, \#)$	$(\#, +)$	q_+							q_e
$(+, \#)$	$(+, +)$	$(+, *)$	$(\#, *)$						q_e
	$(*, \#)$	$(*, *)$	$(*, +)$						q_e

Here are two accepting computations for $e + e * e$:

$$\begin{aligned}
 \#e + e * e\# &\xrightarrow{\#(e)+} \#q_e + e * e\# \xrightarrow{+(e)*} \#q_e + q_e * e\# \xrightarrow{*(e)\#} \#q_e + q_e * q_e\# \xrightarrow{+(*)\#} \\
 \#q_e + q_*\# &\xrightarrow{\#(+)\#} \#q_+\#; \\
 \#e + e * e\# &\xrightarrow{+(e)*} \#e + q_e * e\# \xrightarrow{*(e)\#} \#e + q_e * q_e\# \xrightarrow{\#(e)+} \#q_e + q_e * q_e\# \xrightarrow{+(*)\#} \\
 \#q_e + q_*\# &\xrightarrow{\#(+)\#} \#q_+\#.
 \end{aligned}$$

In general, the syntactic structure is not uniquely determined, since different computations may associate different structures with the same accepted string.

As for grammars, we can formalize a notion of structural ambiguity by using parenthesis automata.

Definition 3. For a chain-driven automaton $\mathcal{A} = \langle \Sigma, Q, C, \delta, F \rangle$, the associated parenthesis automaton is the chain-driven automaton $[\mathcal{A}] = \langle \Sigma \cup \{[,]\}, Q, [C], \delta', F \rangle$ where $[C]$ is the set of chains $a[y]b$ such that $a(y)b \in C$, and δ' is defined by setting $\delta'(a, [\gamma], b) = \delta(a, \gamma, b)$ whenever $\delta(a, \gamma, b) \neq \emptyset$.

A chain-driven automaton \mathcal{A} is structurally ambiguous if $L([\mathcal{A}])$ contains two strings $\mathbf{x}_1 \neq \mathbf{x}_2$ such that $\mathbf{x}_1 \doteq \mathbf{x}_2$.

For instance, consider a variant of the automaton of Example 1, where the context $(+, \#)$ is moved from the second row to the first, and a new body $q_e + q_+$ is added to the second column. This automaton can perform two structurally different computations for the input string $e + e + e$, namely, starting from configuration $\#q_e + q_e + q_e\#$:

$$\begin{aligned}
 \#q_e + q_e + q_e\# &\xrightarrow{\#(+)+} \#q_+ + q_e\# \xrightarrow{\#(+)\#} \#q_+\#, \text{ i.e. where the string is assigned} \\
 &\text{the structure } [[[e] + [e]] + [e]]; \text{ and } \#q_e + q_e + q_e\# \xrightarrow{+(+)\#} \#q_e + q_+\# \xrightarrow{\#(+)\#} \#q_+\#, \\
 &\text{where the structure is } [[e] + [[e] + [e]]].
 \end{aligned}$$

Definition 4. A chain-driven automaton $\mathcal{A} = \langle \Sigma, Q, C, \delta, F \rangle$ is reduced if every chain in C is used in some accepting computation.

W.l.o.g. in what follows we consider only reduced automata.

We are going to see that chain-driven automata recognize CF languages, and can be seen as parsers for CF grammars: states of the automaton correspond to nonterminals of the grammar; any string reduced by the automaton corresponds to the r.h.s of some rule of the grammar, and any state computed by the reduction function corresponds to the nonterminal at the l.h.s of the same rule.

Definition 5. The chain $a\langle y\rangle b$ is a grammatical chain associated with G if there exists a derivation

$$\#T\# \xrightarrow[G]{*} \alpha aAb \beta \xrightarrow[G]{} \alpha a\gamma b \beta \quad (1)$$

with $\hat{\gamma} = y$, $T \in S$. The set of grammatical chains associated with G is denoted by \mathcal{C}_G .

Theorem 1. Chain-driven automata recognize the class of CF languages.

Proof. We prove that the language recognized by any chain-driven automaton can be generated by a grammar, and vice versa. We first need the concept of labeled transition system (LTS), which is a triple (S, Λ, τ) where S is an infinite set of LTS states, Λ is a set of labels, and τ is a set of labelled state transitions (i.e., $\tau \subseteq S \times \Lambda \times S$).

Notice that both grammars and chain-driven automata can be seen as LTS. Formally, a grammar can be seen as the LTS $(V_N \cup \text{OF}(V_N), C, \Leftarrow)$ where the LTS states are all strings in operator forms, the labels are all chains over Σ , and \Leftarrow is defined by setting $\alpha a\gamma b\beta \xleftarrow{c} \alpha aAb\beta$ where $c = a\langle y\rangle b$, $A \rightarrow \gamma$ is a production of G and $\hat{\gamma} = y$. A chain-driven automaton can be seen as the LTS $(\text{OF}(Q), C, \vdash)$ where labels are the chains that drive the automaton, and \vdash is the relation defined by the reduction moves.

Let $G = (V_N, \Sigma, P, S)$. Define the chain-driven automaton $\mathcal{A}_G = \langle \Sigma, Q, \mathcal{C}_G, \delta, F \rangle$ where: $Q = V_N$; $F = S$ is the set of axioms; \mathcal{C}_G is the set of grammatical chains associated with G ; δ is defined by setting $B \in \delta(a, \gamma, b)$ for each production $B \rightarrow \gamma$ such that $a[\hat{\gamma}]b \in \mathcal{C}_G$. Both G and \mathcal{A}_G define the same LTS, except that for G the set of LTS states is $\text{SF}(V_N)$ whereas for \mathcal{A} the LTS states are the configurations of the automaton, i.e., strings $\#\gamma\# \in \text{SF}(V_N)$. In particular, this means that the derivations $T \xrightarrow{*} x$ of G with $T \in S$ are in bijection with the computations $\#x\# \vdash^* \#T\#$ and this implies that $L(\mathcal{A}_G) = L(G)$.

Conversely, let $\mathcal{A} = \langle \Sigma, Q, C, \delta, F \rangle$. Define the grammar $G_{\mathcal{A}} = (V_N, \Sigma, P, S)$ where: $V_N = \Sigma \times Q \times \Sigma$ $S = \{(\#, q, \#) \mid q \in F\}$ P is the set of productions $(a_0, q, a_{n+1}) \rightarrow \gamma$ where $q \in \delta(a_0, \gamma, a_{n+1})$. Both \mathcal{A} and $G_{\mathcal{A}}$ define the same LTS, except that for \mathcal{A} the LTS states are configurations $\#q_0a_1q_1a_2 \dots a_nq_n\#$ (any q_i may be missing), whereas for $G_{\mathcal{A}}$ the LTS states are written in the form $(\#, q_0, a_1)a_1(a_1, q_1, a_2)a_2 \dots a_{n-1}(a_n, q_n, \#)$. In particular, this means that computations $\#x\# \vdash^* \#q\#$ of \mathcal{A} with $q \in F$ are in bijection with the derivations $(\#, q, \#) \xrightarrow{*} x$ of G and this implies that $L(G_{\mathcal{A}}) = L(\mathcal{A})$. Notice that $\mathcal{C}_G = C$. \square

Traditional general CF parsers proceed always left to right and produce a unique representation of the syntax trees associated with the input string; our chain-driven automata, instead, may nondeterministically produce any bottom-up possible traversal of the grammar's trees, as it is illustrated by the parser of Example 1 and by its structurally ambiguous modification. Clearly, \mathcal{A} is structurally unambiguous iff the equivalent grammar $G_{\mathcal{A}}$ defined in the proof of Theorem 1 is structurally unambiguous, and vice versa.

4 Locally Chain-Parsable Languages

The following definitions formalize our intuitive idea of local parsability.

Definition 6. A local chain parser (LCPA) is a chain-driven automaton such that, for every chain $a\langle y\rangle b$, the following condition holds: if $\hat{\gamma} = y$, then every computation $\alpha a\gamma b \beta \vdash^* \#q_F\#$ with $q_F \in F$ can be decomposed as $\alpha a\gamma b \beta \vdash^* \alpha' a\gamma b \beta' \vdash^* \alpha' aqb \beta' \vdash^* \#q_F\#$ with suitable α', β' , and q .

Definition 7. A grammar is locally chain-parsable (LCPG) if, for every grammatical chain $a\langle y\rangle b$, the following condition holds: if $\gamma \hat{=} y$, then each derivation $\#T\# \xRightarrow{*} \alpha a\gamma b\beta$ with $T \in S$ can be decomposed as $\#T\# \xRightarrow{*} \alpha' aAb \beta' \xRightarrow{*} \alpha' a\gamma b \beta'$. A language L is locally chain-parsable (LCPL) if it is generated by a LCPG.

In other terms, for a grammar to be LCPG, we require what follows: for every γ appearing with terminal context (a, b) at the end of some derivation starting from $\#T\#$, γ has to be generated with a single production $A \rightarrow \gamma$ and such a production has to be applied to a string where the nonterminal A already has (a, b) as context.

Theorem 2. A language is LCPL if and only if it is recognized by a LCPA. An LCPA recognizes any string in linear time.

Proof. The statement is a consequence of the fact the both constructions in the proof of Theorem 1 preserve locality properties.

Concerning time complexity, it is well-known that every grammar can be automatically transformed into a structurally equivalent invertible one [8]; thus, if we apply such a procedure to a locally parsable grammar, the corresponding local parser defined by Theorem 1 has a deterministic reduction function δ . It is therefore a simple exercise to derive a traditional deterministic pushdown automaton from a deterministic local parser: the former one simply restricts the set of computations of the latter one to the reverse of the rightmost visit of syntax trees. Thus, LCPL are (strictly) included in DCFL. \square

Example 2. The following grammar G_1 , which generates the same arithmetic expressions recognized by the chain-driven automaton of Example 1, is locally parsable.

$$\begin{array}{ll} E \rightarrow E + T \mid T * F \mid e & F \rightarrow e \\ T \rightarrow T * F \mid e & S = \{E, T, F\} \end{array}$$

In fact, consider a generic derivation such as $\#E\# \Rightarrow \#E + T\# \Rightarrow \#E + T + T\# \Rightarrow \#E + T * F + T\# \xRightarrow{*} \#e + e * F + e\# \Rightarrow \#e + e * e + e\#$. The result of any derivation step is such that each terminal character is enclosed within a context of a pair of terminals which univocally determines the stencil of the last step of the derivation that produced it, independently on the non-terminals involved in the derivation: e.g., every e can only be produced by a rule with

stencil $N \rightarrow e$; the only $*$ in the context $(+, +)$ can only be produced through a rule with stencil $N \rightarrow N * N$; the first $+$ is produced by the rule $E \rightarrow E + T$ in the context $(\#, +)$ but there is no way to produce the second $+$ within any of the contexts $(+, \#)$, $(*, \#)$, $(*, e)$, and (e, e) , by means of an immediate derivation with stencil $N \rightarrow N + N$.

Thus, a possible bottom up parser can always decide which terminal part of any r.h.s. to reduce by only inspecting the terminal parts of any sentential form of length 3 plus its context: if it finds the terminal part $\hat{\alpha}$ of a rule $A \rightarrow \alpha$ within a context where G can generate any β with $\beta \hat{=} \alpha$ through an immediate step of derivation $B \Rightarrow \beta$, then it can reduce the r.h.s to the corresponding l.h.s. with the certainty that the same $\hat{\alpha}$ cannot be obtained as part of a more complex derivation that does not produce it in a single step; notice also that the reduction could be fully deterministic if G were invertible.

On the contrary, the following grammar G_2 , generating only additive expressions, is not locally parsable.

$$\begin{array}{l} X \rightarrow E + X \mid E + E \quad E \rightarrow e \\ Y \rightarrow Y + E \mid E + E \quad S = \{X, Y\} \end{array}$$

The grammatical chains associated with G_2 are $\#(+)\#$, $\#(+)+$, $\#(e)+$, $+ (e)+$, $+ (e)\#$, and $+ (+)\#$. For instance, chain $+ (+)\#$ is obtained by applying rule $X \rightarrow E + E$ in the last step of the following derivation: $\#X\# \xRightarrow{*} \#E + E + X\# \Rightarrow \#E + E + E + E\#$

Now consider the following derivation: $\#Y\# \Rightarrow \#Y + E\# \Rightarrow \#Y + E + E\# \Rightarrow \#E + E + E + E\#$. The factor $\gamma = E + E$ occurs in context $(+, \#)$ but it is not reduced in any step of the derivation. Hence, G_2 is not locally parsable.

Informally, a possible parser, after having reduced all e s to E , would be confronted with the sentential form $\#E + E + E + E\#$ and would not have any indication to decide whether to apply $X \rightarrow E + E$ reducing the last $+$, or $Y \rightarrow E + E$ reducing the first $+$.

4.1 Extended Chains, Conflicts and Decidability of the LCP Property

Both LCPA and LCPG give a unique structure to each string of their respective languages. To formalize this point, we first introduce the notion of extended chain, that generalizes Definition 1.

Definition 8. Structured strings are special well-parenthesized strings over $\Sigma \cup \{[,]\}$, defined recursively as follows:

- $y \in \Sigma^+$ are atomic structured strings;
- if $a_i \in \Sigma$ and $\mathbf{y}_i = \varepsilon$ or $\mathbf{y}_i = [\mathbf{v}_i]$ for some structured strings \mathbf{v}_i , then $\mathbf{y}_0 a_1 \mathbf{y}_1 a_2 \dots a_n \mathbf{y}_n$ is a composed structured string if at least one \mathbf{y}_i is different from ε .

An extended chain (briefly xchain) is a string $\#[\mathbf{y}]\#$ where \mathbf{y} is the body of the xchain.

Any grammar or chain-driven automaton determines a set of xchains which have an important role w.r.t the local parsability property.

Definition 9. Let \mathcal{A} be a chain-driven automaton and G a grammar. An xchain $\#[\mathbf{y}]\#$ is an \mathcal{A} -xchain or a G -xchain, respectively, if there exist γ such that $\tilde{\gamma} = \mathbf{y}$ and

$$\#[\gamma]\# \vdash_{[\mathcal{A}]}^* \#q_F\# \text{ with } q_F \in F \quad \text{or} \quad \#T\# \xrightarrow{[*]}_{[G]} \#[\gamma]\# \text{ with } T \in S.$$

The sets of \mathcal{A} -xchains and G -xchains are denoted respectively by $\mathcal{X}_{\mathcal{A}}$ and \mathcal{X}_G .

Remark 1. If $G_{\mathcal{A}}$ is the grammar equivalent to the chain-driven automaton \mathcal{A} , as defined in the proof of Theorem 1, then $\mathcal{X}_{G_{\mathcal{A}}} = \mathcal{X}_{\mathcal{A}}$; vice versa the chain-driven automaton \mathcal{A}_G equivalent to a grammar G is such that $\mathcal{X}_{\mathcal{A}_G} = \mathcal{X}_G$.

Example 3. Consider grammar G_1 of Example 2. The sentential form $[E + [[e] * [e]]]$ is derived by the associated parenthesis grammar $[G_1]$ with the following derivation $\#E\# \Rightarrow \#[E + T]\# \Rightarrow \#[E + [T * F]]\# \Rightarrow \#[E + [T * [e]]]\# \Rightarrow \#[E + [[e] * [e]]]\#$; hence $\#[+[[e] * [e]]]\#$ is a G_1 -xchain. Other G_1 -xchains are $\#[[+] + [*[e]]]\#, \#[[[e] * [e]] * [e]]\#, \#[+[[* [e]] *]]\#$. Similarly, let \mathcal{A} be the chain-driven automaton of Example 1; both computations for the string $e + e * e$ presented in the same example define the xchain $\#[[e] + [[e] * [e]]]\#$. One can easily guess that \mathcal{A} -xchains are the same as G_1 's ones. Notice also that both G_1 and \mathcal{A} are such that, for each string y they can generate/recognize, there is only one G_1/\mathcal{A} -xchain \mathbf{y} such that $\tilde{\mathbf{y}} = y$.

The next definition introduces the concept of conflict between an xchain and a chain. Intuitively, an xchain c conflicts with a chain $s = a\langle y \rangle b$ if \tilde{c} contains the string ayb but such occurrence of y does not correspond to the body of a “subchain” of s .

Definition 10. An xchain conflicts with a chain $a\langle y \rangle b$ iff it can be decomposed as $\mathbf{x}\mathbf{a}\mathbf{y}\mathbf{b}\mathbf{z}$ where $\tilde{\mathbf{y}} = y$ and $\mathbf{y} \notin [^+\mathbf{y}]^+$. A set \mathcal{X} of xchains and a set C of chains are conflictual iff there is an xchain in \mathcal{X} that conflicts with some chain in C .

Example 4. The xchain $\#[+[+[+[+]]]]\#$ conflicts with the chain $\#\langle + \rangle +$ since the prefix $\#[+[+$ of the xchain projects onto $\# + +$, but the first occurrence of symbol $+$ in the xchain is not bracketed; formally, the definition is satisfied with $\mathbf{x} = \varepsilon$, $\mathbf{y} = [+]$, and $\mathbf{z} = [+[[+]]]\#$. Otherwise, $\#[[[[+]]+]]\#$ does not conflict with $\#\langle + \rangle +$ since when $\# + +$ occurs in the xchain (once, as a prefix), the first occurrence of symbol $+$ is bracketed.

Example 5. By referring again to Example 2, with a little patience it can be verified that the set of G_1 -xchains does not exhibit any conflict with \mathcal{C}_{G_1} , whereas \mathcal{X}_{G_2} and \mathcal{C}_{G_2} are conflictual. We next show that the property of having non-conflictual \mathcal{X}_G and \mathcal{C}_G is decidable for any grammar G (and is supported by an automatic tool.¹) Also, G_1 is locally parsable, whereas G_2 is not. These remarks leads to the main property stated in Theorem 4.

¹ <https://github.com/bzoto/chainsaw>.

Theorem 3. *The fact that \mathcal{X}_G and \mathcal{C}_G are nonconflictual is decidable for every grammar G ; the fact that \mathcal{X}_A and C are nonconflictual is decidable for every automaton A driven by the set of chains C .*

Proof. Let G be (V_N, Σ, P, S) . We first introduce a grammar $G' = (V_N \cup \{T'\}, \Sigma \cup \{[,]\}, P', \{T'\})$, such that $T' \notin V_N$ and

$$P' = \{A \rightarrow [\alpha'] \mid A \rightarrow \alpha \in P, \text{ where } \alpha' = \alpha \text{ or } \alpha' \text{ is obtained from } \alpha \\ \text{by erasing some (or every) nonterminals}\} \cup \{T' \rightarrow \#[T]\# \mid T \in S\}.$$

It is easy to see that G' defines the language of all the G -xchains. For a grammatical chain c , we can define a regular language $R(c) = (\Sigma \cup \{[,]\})^* \cdot a \cdot \neg([\cdot y]^+)$ · $b \cdot (\Sigma \cup \{[,]\})^*$ that is the language of all the possible xchains that conflict with c . Clearly, $R' := \bigcup_{c \in \mathcal{C}_G} R(c)$ is also a regular language. G is conflictual iff $L(G') \cap R' \neq \emptyset$; since $L(G') \cap R'$ is context-free, its emptiness problem is decidable.

The statement for the automaton follows from Theorem 1 and Remark 1. \square

Theorem 4. *A chain-driven automaton A is a local parser if and only if \mathcal{X}_A and the set C of chains driving A are not conflictual. A grammar G is locally parsable if and only if \mathcal{X}_G and \mathcal{C}_G are not conflictual.*

Theorems 4 and 3 imply the following result.

Corollary 1. *The fact that a grammar is LCPG is decidable; the fact that a chain-driven automaton is LCPA is decidable.*

4.2 Basic Properties of Local Chain-Parsable Languages

LCP grammars and parsers associate a unique structure with each generated/accepted string x ; such a structure is represented by an xchain $\#[x]\#$ with $x \doteq x$.

Theorem 5. *LCP grammars and parsers are structurally unambiguous.*

Proof. Both properties can be proved similarly reasoning by contradiction. Assume that \mathcal{A} is structurally ambiguous; then one can show that \mathcal{X}_A and the set of chains that drive \mathcal{A} are conflictual. By Theorem 4 this means that \mathcal{A} is not LCPA. For grammars the same results can be proved by using Corollary 1.

Definition 11. *Two LCPA $\mathcal{A}_1 = (\Sigma, Q_1, C_1, \delta_1, F_1)$ and $\mathcal{A}_2 = (\Sigma, Q_2, C_2, \delta_2, F_2)$ are compatible if $\mathcal{X}_{\mathcal{A}_1} \cup \mathcal{X}_{\mathcal{A}_2}$ and $C_1 \cup C_2$ are not conflictual. Two LCPL L_1 and L_2 are compatible if they are recognized by compatible LCPA.*

Theorem 6. *Let L be LCPL. Then its reversal L^R is LCPL.*

Theorem 7. *Let $\mathcal{A}_1 = (\Sigma, Q_1, C_1, \delta_1, F_1)$ and $\mathcal{A}_2 = (\Sigma, Q_2, C_2, \delta_2, F_2)$ be compatible chain-driven automata recognizing respectively L_1 and L_2 . Then $L_1 \cup L_2$, $L_1 \cap L_2$, and $L_1 \setminus L_2$ are LCPL.*

Proof. W.l.o.g. we may assume that the set of states Q_1 and Q_2 are disjoint. Let $C = C_1 \cup C_2$, and $Q = (Q_1 \cup \{\perp, q_{err}\}) \times (Q_2 \cup \{\perp, q_{err}\})$, with $\perp, q_{err} \notin Q_1 \cup Q_2$. For each strings $\gamma \in \text{OF}(Q)$, say $\gamma = q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n$ with $q_i \in Q \cup \{\varepsilon\}$, define $\gamma_1 = p_0 a_1 p_1 a_2 p_2 \cdots a_n p_n$ where p_i is empty whenever q_i is empty or has \perp as first component, and p_i is the first component of q_i in the other cases; γ_2 is defined symmetrically. Then $\delta(a, \gamma, b)$ is as follows:

- $\delta(a, \gamma, b)$ is the set of all pairs (q_1, q_2) with $q_1 \in \delta_1(a, \gamma_1, b)$ and $q_2 \in \delta_2(a, \gamma_2, b)$, if both $\delta_1(a, \gamma_1, b)$ and $\delta_2(a, \gamma_2, b)$ are nonempty;
- $\delta(a, \gamma, b) = \emptyset$, if both $\delta_1(a, \gamma_1, b)$ and $\delta_2(a, \gamma_2, b)$ are undefined or empty;
- $\delta(a, \gamma, b)$ is the set of all pairs (q_{err}, q_2) with $q_2 \in \delta_2(a, \gamma_2, b)$, if only $\delta_1(a, \gamma_1, b)$ is undefined or empty, and similarly for the symmetric case.

We remark that independent moves of a LCPA (i.e., moves that reduce non-overlapping factors of the input string) can be applied in any order. For $\diamond \in \{\cap, \cup, \setminus\}$, the automaton for $L_1 \diamond L_2$ is given by $(\Sigma, Q, C, \delta, F_\diamond)$, where: $F_\cap = F_1 \times F_2$, $F_\cup = F_1 \times (Q_2 \cup \{q_{err}\}) \cup (Q_1 \cup \{q_{err}\}) \times F_2$, $F_\setminus = F_1 \times (Q \setminus F_2 \cup \{q_{err}\})$. \square

Corollary 2. *The inclusion problem for compatible LCPL is decidable.*

5 LCPL Versus Operator-Precedence and Input-Driven Languages

It is worthwhile to examine the LCPL as an outgrowth of the classical OPL [7], whose knowledge, both theoretically ([6, 9] and for application to parallel parsing [2]), has much progressed in recent years. OPL is a subfamily of DCFL having the local parsability property and characterized by a bottom-up parser that is driven by three binary *precedence relations* between terminals of the grammar, defined as follows. Let α be a r.h.s.:

- a is *equal in precedence* to b ($a \doteq b$), if ab is a factor of $\widehat{\alpha}$;
- a *yields precedence* to b ($a < b$), if $b = \alpha_{\text{first}}$ and some s.f. contains $a\alpha$ as factor;
- a *takes precedence* over b ($a > b$), if $a = \alpha_{\text{last}}$ and some s.f. contains αb as factor.

For a grammar to be an OPG, at most one relation may hold between a pair of terminals. For instance, the relations for G_1 in Example 2 are: $+ > +$, $+ < *$, $+ < e$, $* > +$, $* > *$, $* < e$, $e > +$, $e > *$. The language $L_3 = \{c^n d^n \mid n > 0\}$, generated e.g. by $A \rightarrow cAd \mid cd$, necessarily has the relations $c < c$, $c \doteq d$, $d > d$. Its reversal L_3^R (generated by the mirror grammar) has the symmetric relations $c > c$, $d \doteq c$, $d < d$; therefore the language $L_4 = L_3 \cup L_3^R$ has precedence conflicts such as $c < c$ and $c > c$ and is easily proved not to be an OPL.

Next, after proving that OPL is strictly contained within LCPL, we argue that the extra generative capacity of LCPG has practical value.

Theorem 8. *The OPL family is strictly contained within the LCPL family. Moreover every OPG is locally chain-parsable.*

Proof. To prove that every OPG G is LCP: the grammatical chains C_G are determined by the precedence relations of G , as follows: $a\langle c_1 \cdots c_k \rangle b \in C_G$ iff $a \prec c_1$, $c_i \doteq c_{i+1}$ for every $1 \leq i < k$, and $c_k \succ b$.

Consider now any G 's derivation of type $\#T\# \Rightarrow^* \alpha\gamma b\beta$ with $\widehat{\gamma} = y$, $a\langle y \rangle b \in C_G$; since for each pair of terminals at most one precedence relation holds, it is necessarily $a \prec \gamma_{\text{first}}$, $\gamma_{\text{last}} \succ b$ and \doteq holds between any pair of consecutive terminals in y . Thus, the above derivation must be decomposed into $\#T\# \Rightarrow^* \alpha' a A b \beta' \Rightarrow^* \alpha' a \gamma b \beta' \Rightarrow^* \alpha\alpha\gamma b\beta$: in fact deriving γ in separate steps (e.g.: $\#T\# \Rightarrow^* \alpha' a \gamma_1 \delta \Rightarrow^* \alpha' a \gamma_1 \gamma_2 b \beta' \Rightarrow^* \alpha\alpha\gamma b\beta$, with $\gamma_1 \gamma_2 = \gamma$, γ_1 and $\gamma_2 \neq \varepsilon$) would imply the existence of a \prec or of a \succ relation within γ in conflict with the \doteq relation (in this example $\gamma_{1\text{last}} \succ \gamma_{2\text{first}}$).

The strict inclusion $OPL \subset LCPL$ is witnessed by the previous language, $L_4 = \{c^n d^n \mid n \geq 1\} \cup \{d^n c^n \mid n \geq 1\}$, which is recognized by the obviously local automaton driven by the chains: $\#\langle cd \rangle\#$, $\#\langle dc \rangle\#$, $c\langle cd \rangle d$, $d\langle dc \rangle c$. \square

Useful generative capacity of LCPG. LCPG permit to define relevant syntactic constructs beyond the capacity of OPG. As an argument we present a well-known practical construct: arithmetic expressions containing both unary and binary minus signs, which notoriously introduce precedence conflicts. To keep the example small, the next grammar features only subtraction and multiplication and e as operand, but other operators and parentheses would be straightforward to add. E is the only axiom:

$$\begin{aligned} E &\rightarrow e \mid -e \mid eX \mid -eX \mid E - e \mid E - -e \mid E - eX \mid E - -eX \\ X &\rightarrow *e \mid * - e \mid *eX \mid * - eX \end{aligned}$$

To prove that this grammar has the LCP property, we have used the previously mentioned tool. While traditional precedence parsers are forced to use some trick (like diversifying the two types of minus in the preceding phase of lexical analysis), our LCPG permits a pure syntactic approach.

6 Related Work and Conclusions

In addition to the mentioned relations of our work to the IDL, other classical lines of research present conceptual analogies to be briefly presented, and have somewhat inspired our effort. Brevity forces us to limit explanations and citations.

The NTS languages [5] are defined by the nonterminal separation property. They enjoy the local parsability property in the following sense: if a factor (with terminals and nonterminals) occurs in a sentence as a constituent, i.e., is generated by a nonterminal symbol, then, for every sentence, the same factor can be reduced to the same symbol. NTS languages, however, are not input-driven, because they rely on the presence of nonterminals for localizing the position of a reduction.

Moving to another research area, the local parsability property is remindful of the confluence property of Church-Rosser languages (also called McNaughton

languages): they are defined by string rewriting rules [4, 10]. Such systems, under the length-reducing hypothesis that ensures that the length of reduction chains is not infinite, bear some similarity to our approach. But they are more powerful than ours, because they define also deterministic context-sensitive languages. Moreover, they are not input-driven in any sense, since the rules contain also nonterminal symbols.

To sum up, to our knowledge, our class of automata and grammars differs from all existing, somewhat related, models, either, or both, with respect to the local parsability property and to the input-driven aspects.

This class properly extends the known input-driven classes, preserving important closure properties, and maintains the decidability of the containment problem. This increased generative power can be exploited to define practical languages and to obtain efficient parallel parsers, thus extending the algorithm presented in [3] for OPL and replicating the obtained benefits in terms of time complexity and speed up. Further closure properties, in particular concatenation and Kleene's star, are still to be investigated.

The present paper represents a new step in the long term path towards a general theory of local deterministic parsing.

References

1. Alur, R., Madhusudan, P.: Adding nesting structure to words. *J. ACM* **56**(3), 1–43 (2009)
2. Barengi, A., Crespi Reghizzi, S., Mandrioli, D., Pradella, M.: Parallel parsing of operator precedence grammars. *IPL* **113**, 245–249 (2013)
3. Barengi, A., Crespi Reghizzi, S., Mandrioli, D., Pradella, M.: Parallel parsing made practical. *SCP* (2015). under revision
4. Beaudry, M., Holzer, M., Niemann, G., Otto, F.: McNaughton families of languages. *TCS* **290**(3), 1581–1628 (2003)
5. Boasson, L., Sénizergues, G.: NTS languages are deterministic and congruential. *J. CSS* **31**(3), 332–342 (1985)
6. Crespi Reghizzi, S., Mandrioli, D.: Operator precedence and the visibly pushdown property. *J. CSS* **78**(6), 1837–1867 (2012)
7. Floyd, R.W.: Syntactic analysis and operator precedence. *J. ACM* **10**(3), 316–333 (1963)
8. Harrison, M.A.: *Introduction to Formal Language Theory*. Addison Wesley, New York (1978)
9. Lonati, V., Mandrioli, D., Panella, F., Pradella, M.: Operator precedence languages: Their automata-theoretic and logic characterization. *SICOMP* (2015). to appear
10. McNaughton, R., Narendran, P., Otto, F.: Church-rosser thue systems and formal languages. *J. ACM* **35**(2), 324–344 (1988)
11. Mehlhorn, K.: Pebbling mountain ranges and its application of DCFL-recognition. In: de Bakker, J.W., van Leeuwen, J. (eds.) *ICALP 1980*. LNCS, vol. 85. Springer, Heidelberg (1980)

Classes of Languages Generated by the Kleene Star of a Word

Laure Daviaud¹ and Charles Paperman²(✉)

¹ LIF, UMR7279, CNRS, Aix-Marseille Université, Marseille, France

² Warsaw University, Warsaw, Poland
charles.paperman@gmail.com

Abstract. In this paper, we study the lattice and the Boolean algebra, possibly closed under quotient, generated by the languages of the form u^* , where u is a word. We provide effective equational characterisations of these classes, i.e. one can decide using our descriptions whether a given regular language belongs or not to each of them.

1 Introduction

Equational descriptions of regular languages is a successful and long-standing approach to obtain characterisations of classes of regular languages. One of the first results about equational descriptions is Schützenberger’s theorem [10] on star-free languages. In the case of a variety of regular languages, Reiterman’s theorem [9] guarantees the existence of a characteristic set of profinite equations. This theorem has been extended to several kinds of classes of languages, including lattices and Boolean algebras. The reader could refer to [3, 6] for a more detailed presentation. Let \mathcal{U} be the class of all languages of the form u^* , where u is a word. The aim of this paper is to study the four classes of regular languages \mathcal{L} , \mathcal{B} , $\mathcal{L}q$ and $\mathcal{B}q$ obtained respectively as the closure of \mathcal{U} under the following operations: finite union and finite intersection (lattice operations) for \mathcal{L} , finite union, finite intersection and complement (Boolean operations) for \mathcal{B} , lattice operations and quotients for $\mathcal{L}q$ and Boolean operations and quotients for $\mathcal{B}q$.

Our main result is an equational characterisation for each of these four classes. These equational characterisations being effective, they give as a counterpart the decidability of the membership problem: One can decide whether a given regular language belongs to \mathcal{L} , \mathcal{B} , $\mathcal{L}q$ and $\mathcal{B}q$ respectively. In addition to describing \mathcal{L} , \mathcal{B} , $\mathcal{L}q$ and $\mathcal{B}q$ in terms of equations, our results also provide a general form for the languages belonging to each of these classes.

Motivations. Our motivation for the study of these classes are threefold. First, Restivo suggested a few years ago to characterise the variety of languages generated by the languages of the form u^* , where u is a word. Given that a variety of languages is a class of regular languages closed under Boolean operations,

The second author is supported by WCMCS.

quotients and inverses of morphisms, our result can be viewed as a first step towards the solution of Restivo's problem.

Our second reason for studying these classes was to provide non trivial applications of the equational theory of regular languages as defined by Gehrke, Grigorieff and Pin in [3, 6]. There are indeed plenty of examples of known equational characterisations of varieties of languages, but not so much of classes of languages that are not closed under inverses of morphisms or under quotients.

Our third motivation is rather a long term perspective since it has to do with the (generalised) star-height problem, a long standing open problem on regular languages [7]. It appears that a key step towards this problem would be to characterise the Boolean algebra generated by the languages of the form F^* , where F is a finite language. The case $F = \{u\}$ studied in this paper is certainly a very special case, but it gives an insight into the difficulty of the general problem.

Related Work. A related class is the class of *slender languages* [4, 11], which can be written as a finite union of languages of the form xu^*y , where $x, u, y \in A^*$. The class of *slender or full languages* is a lattice closed under quotients that is therefore characterised by a set of equations. These equations correspond in fact to patterns that cannot be found in any minimal automaton that computes a slender language. In our case, equations provided to characterise classes \mathcal{L} , \mathcal{B} , $\mathcal{L}q$ and $\mathcal{B}q$ can also be seen as forbidden patterns in automata. Then, we deduce normal forms for the languages in \mathcal{L} , \mathcal{B} , $\mathcal{L}q$ and $\mathcal{B}q$.

Organization of the Paper. Section 2 gives classical definitions and properties about the algebraic automata theory and profinite semigroups. Section 3 is dedicated to the study of the syntactic monoid of u^* for a given word u . In particular, we present useful algebraic properties of the syntactic monoid of u^* . Section 4 presents equational theory of regular languages: it first gives classical results, then presents the equations satisfied by u^* , and finally gives the characterisations of \mathcal{L} , $\mathcal{L}q$ and $\mathcal{B}q$. The study of \mathcal{B} is much more intricate and involves specific tools that are given in Sect. 5. Finally, Sect. 6 presents decidability issues. Sections 2 to 6 deal with alphabet with at least two letters. The case of a unary alphabet is simpler and derives from the two-letter case. It is treated in Sect. 7.

Notations. We denote by A a finite alphabet with at least two letters, by A^* the set of words on A , by 1 the empty word and by $|u|$ the length of a word u .

2 Recognisability and the Profinite Monoid

In this section, we introduce the definitions of recognisability by monoids and of profinite monoid. For more details, the reader could refer to [2].

Monoids and Recognisability. A monoid M is a set equipped with a binary associative operation with a neutral element denoted by 1 . The product of x and y is denoted by xy . An element e of M is idempotent if $e^2 = e$. An element

$0 \in M$ is a zero of M if for all $x \in M$, $0x = x0 = 0$. Given two monoids M and N , $\varphi : M \rightarrow N$ is a morphism if for all $x, y \in M$, $\varphi(xy) = \varphi(x)\varphi(y)$ and $\varphi(1) = 1$.

In a finite monoid, every element has an idempotent power: for all $x \in M$, there is $n_x \in \mathbb{N} - \{0\}$ such that x^{n_x} is idempotent. The smallest n_x satisfying this property is called the *index* of x . Moreover, there is an integer $n \neq 0$ such that for all $x \in M$, x^n is idempotent. For instance, one could take the product of the n_x . The smallest integer satisfying this property is called the *index* of the monoid and is denoted by ω . Thus, x^ω is the unique idempotent in the subsemigroup generated by x .

Given a monoid M and a morphism $\varphi : A^* \rightarrow M$, a language L is said to be *recognised* by (M, φ) if there is $P \subseteq M$ such that $L = \varphi^{-1}(P)$. The language L is said to be recognised by M if there is φ such that (M, φ) recognises L . A language is regular if and only if it is recognised by a finite monoid. Moreover, the smallest monoid that recognises a regular language L is unique up to isomorphism and is called the *syntactic monoid* of L . The associated morphism φ is called the *syntactic morphism* and $\varphi(L)$ is called the *syntactic image* of L . Furthermore, for each word u , we call $\varphi(u)$ the syntactic image of u with respect to L . The syntactic monoid of a regular language can be computed as it is the transition monoid of the minimal (deterministic) automaton of L .

Free Profinite Monoid. Given two words u and v , a monoid M separates u and v if there is a morphism $\varphi : A^* \rightarrow M$ such that $\varphi(u) \neq \varphi(v)$. If $u \neq v$, there is a finite monoid that separates u and v . A distance d can be defined on A^* as follows: $d(u, u) = 0$ and if $u \neq v$, $d(u, v) = 2^{-n}$ where n is the smallest size of a monoid that separates u and v . Moreover this distance is ultrametric.

Every finite monoid is seen as a metric space equipped with the distance $d(x, y) = 1$ if $x \neq y$ and $d(x, y) = 0$ otherwise. This implies that every morphism from A^* to a finite monoid is a uniformly continuous function.

We briefly recall some useful definitions and results on the *free profinite monoid*. We refer to [2] for an extended presentation of this subject. The free profinite monoid of A^* , denoted by $\widehat{A^*}$ can be defined as the completion for the distance d of A^* . It is a compact space such that A^* is a dense subset of $\widehat{A^*}$. Its elements are called *profinite words*. It is known that a language L is regular if and only if \overline{L} is open and closed in $\widehat{A^*}$, where \overline{L} is the topological closure of L in $\widehat{A^*}$.

Finally, every morphism from A^* to some finite monoid M can be uniquely extended to a uniformly continuous morphism from $\widehat{A^*}$ to M . By abuse of notation, a morphism and its extension will be denoted by the same symbol.

The two following examples are profinite words that are not finite words and that will be intensively used in the remainder of the paper.

Example 1. (Idempotent power). Given a word $u \in A^*$, the sequence $(u^n)_n$ converges in $\widehat{A^*}$. Its limit is denoted by u^ω . Given a finite monoid M and a morphism $\varphi : \widehat{A^*} \rightarrow M$, $\varphi(u^\omega) = \varphi(u)^\omega$.

Example 2. (Zero [1, 8]). Let A be an alphabet with at least two letters and fix a total order on it. Let $(u_n)_n$ be the sequence of all words ordered by the induced shortlex order. We set: $v_0 = u_0$ and for all $n \in \mathbb{N}$, $v_{n+1} = (v_n u_{n+1} v_n)^{(n+1)!}$. The sequence $(v_n)_n$ converges in \widehat{A}^* and we denote by ρ_A its limit. Given a finite monoid M and a morphism $\varphi : \widehat{A}^* \rightarrow M$, if M has a zero then $\varphi(\rho_A) = 0$.

3 The Languages u^*

As mentioned in the introduction, our goal is to describe classes generated by the languages u^* . We will see in Sect. 4 that proving the correctness of such characterisations requires a precise description of the structure of the syntactic monoid of a given language u^* and particularly of its idempotents.

Therefore, this section addresses this study by exhibiting some properties of the syntactic monoid of u^* . Let us introduce two notions useful to study the languages of the form u^* . A word u is said to be *primitive* if for all words v and all integers n , the condition $u = v^n$ implies $n = 1$ and $v = u$. A word v is said to be a *conjugate* of u if there are words u_1, u_2 such that $u = u_1 u_2$ and $v = u_2 u_1$. The study of the syntactic monoid of u^* highly depends on the fact whether u is primitive or not. Without loss of generality, we consider now the studied language to be of the form $(u^m)^*$ for u a primitive word and m a positive integer.

In the syntactic monoid of $(u^m)^*$, there is a zero that is the syntactic image of words that cannot be completed into a word of $(u^m)^*$. Idempotent elements are exactly this zero, the neutral element and the syntactic images of the conjugates of u^m . Thus there are $|u| + 2$ idempotents. Moreover, if the idempotent power of a syntactic image of a word is not zero, then this word has to be a power of a conjugate of u . Finally, the index of the syntactic images of u and of its conjugates is m . All these properties are instantiated in Example 3.

Example 3. We show in Fig. 1 the minimal deterministic automaton and monoid representation of the language $(aab)^*$. The elements in boxes are the elements

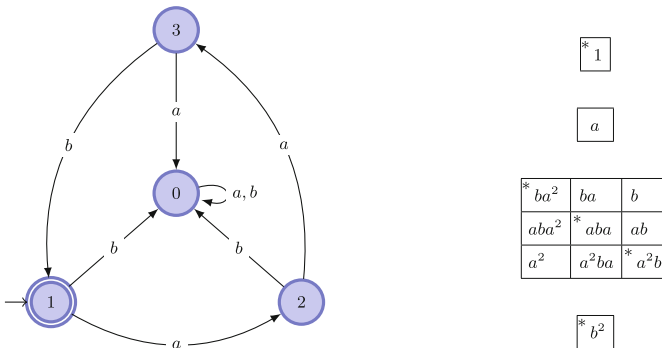


Fig. 1. Minimal deterministic automaton and monoid representation of $(aab)^*$

of the syntactic monoid of $(aab)^*$. An element has a star in its box if it is idempotent. The conjugates of aab are aab , aba and baa . Finally, the syntactic image of b^2 is a zero of the monoid.

4 Equational Characterisations of \mathcal{L} , $\mathcal{L}q$ and $\mathcal{B}q$

This section covers the equational theory of regular languages. First, Sect. 4.1 presents known results about equations. Then Sect. 4.2 applies this theory to the study of \mathcal{L} , $\mathcal{L}q$ and $\mathcal{B}q$, by giving equations that characterise them.

4.1 Equational Characterisations of Algebraic Structures of Regular Languages

A *lattice* (resp. a *Boolean algebra*) of languages of A^* is a class of languages containing the empty language \emptyset , the full language A^* and which is closed under finite union and finite intersection (resp. finite union, finite intersection and complement). A class of languages \mathcal{L} is *closed under quotients* if for all $L \in \mathcal{L}$, for all $u \in A^*$, $u^{-1}L$ and Lu^{-1} belong to \mathcal{L} . Recall that $u^{-1}L = \{v \mid uv \in L\}$ and $Lu^{-1} = \{v \mid vu \in L\}$. Let u and v be two profinite words. A language $L \subseteq A^*$ satisfies the equation $u \rightarrow v$ if the condition $u \in \bar{L}$ implies $v \in \bar{L}$. It satisfies $u \leq v$ if for all words x, y , $xuy \in \bar{L}$ implies $xvy \in \bar{L}$. The notation $u \leftrightarrow v$ is a shortcut for $u \rightarrow v$ and $v \rightarrow u$ and similarly $u = v$ is a shortcut for $u \leq v$ and $v \leq u$. Observe that given a regular language L and its syntactic morphism $\varphi : A^* \rightarrow M$, the language L satisfies $u = v$ if and only if $\varphi(u) = \varphi(v)$ in M . A class of languages \mathcal{L} is defined by a set of equations E if the following equivalence holds: $L \in \mathcal{L}$ if and only if L satisfies all the equations in E .

The kind of equations used to describe a class of languages is strongly related to its closure operations. The two following propositions formalise this statement.

Proposition 1. (Theorem 5.2 [3]). *A class of regular languages is defined by a set of equations of the form $u \rightarrow v$ (resp. $u \leftrightarrow v$) if and only if it is a lattice (resp. a Boolean algebra) of regular languages.*

Proposition 2. (Theorem 7.2 [3]). *A class of regular languages is defined by a set of equations of the form $u \leq v$ (resp. $u = v$) if and only if it is a lattice (resp. a Boolean algebra) of regular languages closed under quotients.*

Equations with zero. The existence of a zero in a syntactic monoid is given by the equations:

$$\rho_A x = x \rho_A = \rho_A$$

If these equations are satisfied, we will use the notation 0 instead of ρ_A . For example the set of equations:

$$\left\{ \begin{array}{l} \rho_A \leq x \\ \rho_A x = x \rho_A = \rho_A \end{array} \right. \quad \text{is replaced by } 0 \leq x$$

The zero has been used to describe several classes of languages. For instance, the equations $0 \leq x$ for $x \in A^*$ describe exactly the so called *nonsense* languages. Another example is the class of slender or full languages defined in the introduction [4, 11]. This class of languages is a lattice closed under quotients; it is described by the following equations:

$$0 \leq x \text{ for } x \in A^*$$

$$x^\omega uy^\omega = 0 \text{ for } x, y \in A^+, u \in A^* \text{ and } i(uy) \neq i(x)$$

where $i(v)$ is the first letter of v for any $v \in A^+$ [5].

4.2 Characterisations of \mathcal{L} , $\mathcal{L}q$ and $\mathcal{B}q$

We give here a list of equations used in the study of \mathcal{L} , $\mathcal{L}q$, $\mathcal{B}q$ and \mathcal{B} . The proofs of the characterisations of these classes by some sets of equations are made in two steps. We first verify that the equations are correct and then check for their completeness. For the first step, it is sufficient to prove that for all words u , the language u^* satisfies the set of equations. From the nature of the equations (\rightarrow , \leftrightarrow , \leq , $=$), we then obtain directly that the whole lattice, Boolean algebra and their closure under quotients satisfy the given set of equations. This step of correctness can be derived from the structure of the languages of the form u^* , presented in Sect. 3. The second step is to prove that only the languages in the desired structures satisfy the set of equations. This step is more intricate since it requires a full understanding of the combinatorics of the classes we consider. First we define the two following languages:

$$P_u = \bigcup_{p \text{ prefix of } u} u^*p \quad \text{and} \quad S_u = \bigcup_{s \text{ suffix of } u} su^*$$

The equations:

$$x^\omega y^\omega = 0 \text{ for } x, y \in A^* \text{ such that } xy \neq yx \tag{E_1}$$

$$x^\omega y = 0 \text{ for } x, y \in A^* \text{ such that } y \notin P_x \tag{E_2}$$

$$yx^\omega = 0 \text{ for } x, y \in A^* \text{ such that } y \notin S_x \tag{E_3}$$

$$x^\omega \leq 1 \text{ for } x \in A^* \tag{E_4}$$

$$0 \leq 1 \tag{E_5}$$

$$x^\ell \leftrightarrow x^{\omega+\ell} \text{ for } x \in A^*, \ell > 0 \tag{E_6}$$

$$x^\omega \rightarrow 1 \text{ for } x \in A^* \tag{E_7}$$

$$x \rightarrow x^\ell \text{ for } x \in A^*, \ell > 0 \tag{E_8}$$

Some equations are clearly satisfied by u^* such as equations (E_8) and (E_7) . Indeed, if $v \in u^*$ then for all ℓ , v^ℓ is also a power of u and belongs to u^* (E_8) . Similarly, 1 always belongs to u^* (E_7) . Proving that u^* satisfies the other equation is more difficult and requires to analyse the structure of its syntactic monoid. In particular, the role of the idempotents is important. The following theorem gives the equational characterisations of $\mathcal{B}q$, $\mathcal{L}q$ and \mathcal{L} .

Theorem 1. *Over a finite alphabet with at least two letters:*

1. *The class $\mathcal{B}q$ is defined by equations (E_1) , (E_2) and (E_3) .*
2. *The class $\mathcal{L}q$ is defined by equations (E_1) , (E_2) , (E_3) and (E_4) .*
3. *The class \mathcal{L} is defined by equations (E_1) , (E_4) and (E_8) .*

To prove these characterisations we introduce a normal form for the languages in $\mathcal{B}q$, $\mathcal{L}q$ and \mathcal{L} . More precisely, we prove that a language that satisfies the sets of equations can be written in a normal form. Finally, normal forms imply membership in the classes $\mathcal{B}q$, $\mathcal{L}q$ or \mathcal{L} . We now sketch briefly the proofs.

We start with the most general class $\mathcal{B}q$ and then we restrict to the classes $\mathcal{L}q$ and \mathcal{L} by adding sets of equations in the equational characterisation. Hence, let us start with $\mathcal{B}q$. First, we remark that the finite languages are in $\mathcal{B}q$, as for instance, the language $\{aab\}$. Indeed, $\{aab\} = a^{-1}(aaab)^* \cap (aab)^*$. Given a word u , and a non-negative integer r , we denote by $u^{\geq r}$ the language u^*u^r . Since this language can be rewritten as $u^* - \{1, u, \dots, u^{r-1}\}$, it belongs to $\mathcal{B}q$. Similarly, by using the closure by quotient we capture the languages $u^{\geq r}p$ and $su^{\geq r}$ where p (resp. s) is a prefix (resp. a suffix) of u . Finally, the following normal form fully characterises the class $\mathcal{B}q$: if L is a nonfull language in $\mathcal{B}q$, then L can be written as

$$\left(\bigcup_{i=1}^k u_i^{\geq r_i} p_i \right) \cup F \text{ or } \left(\left(\bigcup_{i=1}^k u_i^{\geq r_i} p_i \right) \cup F \right)^c$$

where $(u_i)_{i=1\dots k}$ and F are finite sets of words, p_i is a prefix of u_i and $(r_i)_{i=1\dots k}$ are integers. We have sketched the proof that all the languages that can be written in this normal form are in $\mathcal{B}q$. The difficult part is to prove that every regular language that satisfies the equations can be written in the normal form.

We can achieve the reduction from $\mathcal{B}q$ to $\mathcal{L}q$, that is removing the closure by complement, by adding the set of equations (E_4) in the equational characterisation. Furthermore, we obtain that the normal form is a restriction of the previous one: if $L \in \mathcal{L}q$ is nonfull, then

$$L = \left(\bigcup_{i=1}^k u_i^* p_i \right) \cup F$$

Remark 1. The proof is constructive: assuming that a language L satisfies the set of equations, one can compute the words and the integers giving the normal form.

Example 4. The language A^*aaA^* is not in $\mathcal{B}q$. Indeed, the first equation is not satisfied since the syntactic image of the words ab and b are idempotents, but the syntactic image of abb is not syntactically equal to 0. However, the language $A^*(aa + bb)A^*$ satisfies the three sets of equations and is therefore in $\mathcal{B}q$ but not in \mathcal{L} since the set of equations (E_4) is not satisfied: the syntactic image of aa is 0, and by equation (E_4) , $0 \leq 1$, so 1 should be in the language but that is not the case. We can even give the normal form of this language:

$$A^*(aa + bb)A^* = ((ab)^* \cup (ab)^*a \cup (ba)^* \cup (ba)^*b)^c$$

In order to study \mathcal{L} and \mathcal{B} , we have to remove the “closure under quotients” from the characterisations above. We deal with these cases by introducing an intermediate Boolean algebra (resp. lattice) denoted by $\tilde{\mathcal{B}}$ (resp. $\tilde{\mathcal{L}}$). The latter classes are generated by the following languages, which correspond to a certain form of quotients:

$$\tilde{\mathcal{U}} = \{(u^m)^* u^r \mid u \in A^*, m > 0, 0 \leq r < m\}$$

The study of these two classes is an intermediate step since:

$$\mathcal{B} \subseteq \tilde{\mathcal{B}} \subseteq \mathcal{B}q \quad \text{and} \quad \mathcal{L} \subseteq \tilde{\mathcal{L}} \subseteq \mathcal{L}q$$

Proposition 3. *Over a finite alphabet with at least two letters:*

1. *The class $\tilde{\mathcal{B}}$ is defined by equations (E_1) and (E_6) .*
2. *The class $\tilde{\mathcal{L}}$ is defined by equations (E_1) , (E_6) , (E_5) and (E_7) .*

From this proposition, we can see that the language presented in Example 4 $A^*(aa + bb)A^*$ is not in $\tilde{\mathcal{B}}$, and therefore it is neither in \mathcal{L} nor in \mathcal{B} , since the equation (E_6) is not satisfied. Indeed, it is sufficient to consider the word aba , and to remark that $(aba)^2 aba \in A^*(aa + bb)A^*$ but $aba \notin A^*(aa + bb)A^*$. As for the preceding cases, the languages in $\tilde{\mathcal{B}}$ and $\tilde{\mathcal{L}}$ can be written in a normal form: if L is a nonfull language in $\tilde{\mathcal{B}}$, then

$$L \cup \{1\} = \bigcup_{i=1}^k (u_i^m)^* u_i^{r_i} \quad \text{or} \quad L - \{1\} = \left(\bigcup_{i=1}^k (u_i^m)^* u_i^{r_i} \right)^c$$

Similarly, if L is a nonfull language in $\tilde{\mathcal{L}}$, then $L = \bigcup_{i=1}^k (u_i^m)^* u_i^{r_i}$ where $(u_i)_{i=1, \dots, k}$ are words and $m, (r_i)_{i=1, \dots, k}$ are integers. Finally, we can characterise the classes \mathcal{L} and \mathcal{B} by restricting the set of integers r_i that can be obtained in the normal form of $\tilde{\mathcal{L}}$ and $\tilde{\mathcal{B}}$. Regarding \mathcal{L} , one can prove that the only possible choice for r_i is 0. Thus, a nonfull language L in \mathcal{L} is of the form $L = \bigcup_{i=1}^k u_i^*$. Unlike the class \mathcal{L} , the case of \mathcal{B} can not be deduced directly from the case of $\tilde{\mathcal{B}}$ and it is much more complicated. It is the subject of the next section.

5 The Case of the Boolean Algebra \mathcal{B}

We enter here the most intricate part of the description of the classes generated by the languages of the form u^* . The idea is to restrict the possible integers r_i we can obtain in the description of $\tilde{\mathcal{B}}$. For that, we will define equivalence relations over the integers. Once this will be done, the main difficulty will be to translate properties over integers into profinite equations. In order to do that, we will introduce profinite numbers. This issue is addressed in Sect. 5.1 that first defines which sets of integers are allowed for the r_i and then translates it into equations. Finally, Sect. 5.2 aggregates all these notions to give the characterisation of \mathcal{B} .

5.1 Equivalence Classes Over \mathbb{N} and Profinite Numbers

Let m be an integer, and r and s be in $\{0, \dots, m - 1\}$, let us define $r \equiv_m s$ if and only if $\gcd(r, m) = \gcd(s, m)$. Remark that \equiv_m is an equivalence relation. Intuitively, a language in \mathcal{B} with m as the index of its syntactic monoid, will not be able to separate two integers that are equivalent with respect to \equiv_m . More precisely, let L be a language in \mathcal{B} with m as the index of its syntactic monoid and $r \equiv_m s$. Then for all words u and for all k, k' , we have $u^{km+r} \in L$ if and only if $u^{k'm+s} \in L$.

Example 5. We introduce the language $L = (a^2)^* - (a^6)^*$. This language is, by definition, in \mathcal{B} . The index of its syntactic monoid is 6. Classes for \equiv_6 are $\{1, 5\}$, $\{2, 4\}$ and $\{3\}$. Thus, L cannot separate a word in $(a^6)^*a^2$ from a word in $(a^6)^*a^4$. Therefore, since $(a^6)^*a^2$ is in L , $(a^6)^*a^4$ is also in L . Since L belongs to \mathcal{B} , it also belongs to $\widehat{\mathcal{B}}$ and we have a convenient normal form given by Proposition 3:

$$L = (a^2)^* - (a^6)^* = (a^6)^*a^2 \cup (a^6)^*a^4.$$

The equivalence relation \equiv_m allows to give the form of the languages in \mathcal{B} . The next step is to translate it in terms of equations. The difficulty comes from the fact that \equiv_m depends on the parameter m that represents the index of the syntactic monoid of a given language. So, this cannot be directly translated into a set of equations that are supposed to not depend on a specific language.

Profinite Numbers. Consider a one-letter alphabet $B = \{a\}$ and the profinite monoid \widehat{B}^* . There is an isomorphism from B^* to \mathbb{N} that associates a word to its length. Then there is a unique set $\widehat{\mathbb{N}}$ and a unique isomorphism $\psi : \widehat{B}^* \rightarrow \widehat{\mathbb{N}}$ such that $\mathbb{N} \subseteq \widehat{\mathbb{N}}$ and $\widehat{\psi}$ coincides with ψ on \mathbb{N} . Elements of $\widehat{\mathbb{N}}$ are called *profinite numbers*. They are limits of sequences of integers, in the sense of the topology of the set of words on a one-letter alphabet. Given a word u , and a profinite number α , u^α corresponds to the profinite word that is the limit of the words u^{α_n} where $(\alpha_n)_n$ is a sequence of integers converging to α .

Let $\mathcal{P} = \{p_1 < p_2 < \dots < p_n < \dots\}$ be a cofinite sequence of prime numbers. That is, a sequence of prime numbers such that only a finite number of prime numbers are not used in the sequence. Consider the sequence defined by $z_n^{\mathcal{P}} = (p_1 \cdots p_n)^{n!}$. The sequence $(z_n^{\mathcal{P}})_{n>0}$ is converging in $\widehat{\mathbb{N}}$ and we denote by $z^{\mathcal{P}}$ its limit.

We can give now the last set of equations needed to characterise \mathcal{B} and that conveys the notion of equivalence over \mathbb{N} defined above. Denote by Γ the set of pairs of profinite numbers $(dz^{\mathcal{P}}, dpz^{\mathcal{P}})$ satisfying the three following conditions:

- \mathcal{P} is a cofinite sequence of prime numbers,
- $p \in \mathcal{P}$,
- if q divides d then $q \notin \mathcal{P}$.

Let us define the set of equations (E_9) by:

$$x^\alpha \leftrightarrow x^\beta \text{ for all } (\alpha, \beta) \in \Gamma \tag{E_9}$$

5.2 Characterisation of \mathcal{B}

The following result combines the notions given in Sect. 5.1 and characterises the class \mathcal{B} .

Theorem 2. *Over a finite alphabet with at least two letters, the class \mathcal{B} is defined by equations (E_1) , (E_6) and (E_9) .*

Sketch of the Proof. Firstly, we prove that u^* satisfies (E_1) , (E_6) and (E_9) . For (E_9) , essentially, $dpz^{\mathcal{P}}$ is a multiple of $dz^{\mathcal{P}}$ so u^* satisfies $x^{dz^{\mathcal{P}}} \rightarrow x^{dpz^{\mathcal{P}}}$. Conversely, thanks to the definition of Γ , for n large enough, $dz_{n+1}^{\mathcal{P}}$ is a multiple of $dpz_n^{\mathcal{P}}$ and thus u^* satisfies $x^{dpz_n^{\mathcal{P}}} \rightarrow x^{dz_n^{\mathcal{P}}}$.

The reverse implication is proved in two steps. First, we prove that if a nonfull language L satisfies (E_1) , (E_6) and (E_9) , then just like for the other classes, it has a normal form:

$$L \cup \{1\} = \bigcup_{i=1}^k \bigcup_{r \in S_i} (u_i^m)^* u_i^r \quad \text{or} \quad (L - \{1\})^c = \bigcup_{i=1}^k \bigcup_{r \in S_i} (u_i^m)^* u_i^r$$

where m is an integer, $(u_i)_{i=1, \dots, k}$ is a finite set of words, and S_i is an equivalence class of $\equiv_{m\sim}$. We start by using the first part of Proposition 3 to prove that L belongs to \mathcal{B} . So L can be written as:

$$L \cup \{1\} = \bigcup_{i=1}^k (u_i^m)^* u_i^{r_i} \quad \text{or} \quad L - \{1\} = \left(\bigcup_{i=1}^k (u_i^m)^* u_i^{r_i} \right)^c$$

We prove that for all $t \equiv_m r$, u^r belongs to L if and only if u^t belongs to L . The idea is the following: Let φ be the syntactic morphism of L , consider any cofinite sequence of prime numbers \mathcal{P} . If all the prime divisors of m are in \mathcal{P} , then for all n large enough, m divides $z_n^{\mathcal{P}}$ and thus for all words x , $\varphi(x^{dz^{\mathcal{P}}}) = \varphi(x^\omega) = \varphi(x^{dpz^{\mathcal{P}}})$. If none of the prime divisors of m is in \mathcal{P} , then for all n large enough, $z_n^{\mathcal{P}}$ is of the form $km + 1$. Then $\varphi(x^{dz^{\mathcal{P}}}) = \varphi(x^{\omega+d})$ and $\varphi(x^{dpz^{\mathcal{P}}}) = \varphi(x^{\omega+dp})$. Finally, d and dp under the conditions that define the set Γ , represent integers in the same equivalence class with respect to m that are then linked by (E_9) . Other situations are combinations of these two.

Once we have the normal form for L , what is left is to prove that a language that can be written in this normal form belongs to \mathcal{B} . This is done by proving that:

$$\bigcup_{p \in \bar{r}^m} (u^m)^* u^p = (u^d)^* - \bigcup_{\substack{k \text{ s.t.} \\ 0 \leq k \leq m \\ \gcd(k, \frac{m}{d}) \neq 1}} (u^{kd})^*$$

where \bar{r}^m is the equivalence class of r for \equiv_m and $d = \gcd(m, r)$.

6 Decidability

The characterisations that are given in Theorems 1 and 2 yield as a counterpart the decidability of the classes $\mathcal{B}q$, $\mathcal{L}q$, \mathcal{L} and \mathcal{B} : given a regular language L , one can decide if L belongs to said classes. Every single equation is effectively testable. The main issue is to test an infinite set of equations in finite time. The idea is to test the equations in the syntactic monoid of L that is finite and thus test a finite number of equations. The first step is to compute M , the syntactic monoid of L , m its index, φ the syntactic morphism and P , the syntactic image of L . They are all computable from the minimal automaton of L . Then, it is sufficient to check if the sets of equations are satisfied directly in M and P , which are finite. More precisely:

(E_4): For all $x, y, z \in M$, $yx^mz \in P \Rightarrow yz \in P$

(E_5): Particular case of (E_4)

(E_6): For all $x \in M$, for all $0 < \ell < m$, $x^\ell \in P \Leftrightarrow x^{m+\ell} \in P$

(E_7): Particular case of (E_4)

(E_8): For all $x \in M$, for all $0 < \ell \leq 2m$, $x \in P \Rightarrow x^\ell \in P$

(E_9): Thanks to the notion of equivalence classes given in Sect. 5.1, testing equations in (E_9) is the same as testing that for all $x \in M$, for all $0 \leq r, s < m$ such that $r \equiv_m s$, $x^r \in P \Leftrightarrow x^s \in P$.

It is much more difficult to translate sets of equations (E_1), (E_2) and (E_3) in M since conditions " $xy \neq yx$ ", " $y \notin P_x$ " and " $y \notin S_x$ " cannot be translated directly in M .

(E_1): Consider $x, y \in M$ such that $x^m y^m \neq 0$. One has to check that for all words $u \in \varphi^{-1}(x)$, $v \in \varphi^{-1}(y)$, $uv = vu$. This problem is decidable.

(E_2): Consider $x, y \in M$ such that $x^m y \neq 0$. One has to check that for all words $u \in \varphi^{-1}(x)$, $v \in \varphi^{-1}(y)$, $v \in P_u$. This problem is decidable.

(E_3): Same as (E_2)

7 The Case of a Unary Alphabet

This section summarises results for a unary alphabet. In this case, the syntactic monoid of a language of the form $(a^k)^*$ has no zero and even more the construction of ρ_A , given in [1, 8] for larger alphabets, does not make sense for a singleton alphabet. But using the fact that a regular language on the alphabet $A = \{a\}$ is a finite union of languages of the form $(a^q)^* a^p$ for non negative integers p and q , we can derive from proofs made for the general case that $\mathcal{B}q$ is the set of all regular languages. The set $\mathcal{L}q$ is the set of the languages that are finite unions of languages of the form $(a^q)^* a^p$ with $p < q$ and is characterised by (E_4). The set \mathcal{L} is the set of the languages that are finite unions of languages of the form $(a^q)^*$ and is characterised by (E_4) and (E_8). Finally, \mathcal{B} is characterised by (E_6) and (E_9).

8 Conclusion

This paper offers an equational description of the lattice, Boolean algebra and their closure under quotients generated by the languages of the form u^* . These descriptions illustrate the power of the topological framework introduced by [3]. In particular, it gives us tools to describe in an effective way these classes of languages.

A lot of combinatorial phenomena have been understood and analysed to obtain these results. The next step could be to investigate either the case of the classes of languages generated by F^* where F is a finite set of words, or the case of the classes generated by $u_1^*u_2^*\dots u_k^*$ with u_1, \dots, u_k some finite words. Each of these questions are interesting to have a better understanding of the phenomena that appear in the study of the variety generated by the languages u^* and of the generalised star-height problem.

References

1. Almeida, J., Volkov, M.V.: Profinite identities for finite semigroups whose subgroups belong to a given pseudovariety. *J. Algebra Appl.* **2**(2), 137–163 (2003)
2. Almeida, J., Weil, P.: Relatively free profinite monoids: an introduction and examples. In: *Semigroups, Formal Languages and Groups (York, 1993)*, of NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci., vol. 466, pp. 73–117. Kluwer Acad. Publ., Dordrecht (1995)
3. Gehrke, M., Grigorieff, S., Pin, J.É.: Duality and equational theory of regular languages. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008, Part II*. LNCS, vol. 5126, pp. 246–257. Springer, Heidelberg (2008)
4. Honkala, J.: On slender languages. In: Paun, B., Rozenberg, G., Salomaa, A. (eds.) *Current Trends in Theoretical Computer Science*, pp. 708–716. World Scientific Publishing, River Edge (2001)
5. Pin, J.É.: Mathematical foundations of automata theory. <http://www.liafa.jussieu.fr/~jep/PDF/MPRI/MPRI.pdf>
6. Pin, J.É.: Profinite methods in automata theory. In: Albers, S., Marion, J.-Y. (eds.) *26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009)*, pp. 31–50. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2009)
7. Pin, J.É., Straubing, H., Thérien, D.: Some results on the generalized star-height problem. *Inf. Comput.* **101**, 219–250 (1992)
8. Reilly, N.R., Zhang, S.: Decomposition of the lattice of pseudovarieties of finite semigroups induced by bands. *Algebra Univers.* **44**(3–4), 217–239 (2000)
9. Reiterman, J.: The Birkhoff theorem for finite algebras. *Algebra Univers.* **14**(1), 1–10 (1982)
10. Schützenberger, M.P.: On finite monoids having only trivial subgroups. *Inf. Control* **8**, 190–194 (1965)
11. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) *Handbook of Language Theory*, vol. 1, chap. 2, pp. 679–746. Springer, Heidelberg (1997)

Relating Paths in Transition Systems: The Fall of the Modal Mu-Calculus

Cătălin Dima¹, Bastien Maubert², and Sophie Pinchinat³✉

¹ Université Paris Est, LACL, UPEC, Créteil, France
dima@u-pec.fr

² LORIA - CNRS / Université de Lorraine, Nancy, France
bastien.maubert@gmail.com

³ IRISA, Université de Rennes 1, Rennes, France
sophie.pinchinat@irisa.fr

Abstract. We revisit Janin and Walukiewicz’s classic result on the expressive completeness of the modal mu-calculus w.r.t. MSO , when transition systems are equipped with a binary relation over paths. We obtain two natural extensions of MSO and the mu-calculus: *MSO with path relation* and the *jumping mu-calculus*. While “bounded-memory” binary relations bring about no extra expressivity to either of the two logics, “unbounded-memory” binary relations make the bisimulation-invariant fragment of MSO with path relation more expressive than the jumping mu-calculus: the existence of winning strategies in games with imperfect-information inhabits the gap.

1 Introduction

Monadic second-order logic (MSO) is a standard for comparing expressiveness of other logics of programs. Ground-breaking expressiveness results on MSO were obtained first on “freely-generated” structures (words, trees, tree-like structures, etc.) [26, 29], then on “non-free” structures like grids [18] or infinite graphs generated by regularity-preserving transformations [8, 10]. Some attention has also been brought to the study of enrichments of MSO with unary predicate symbols or with the “equal level” binary predicate (MSO^{eq}) [11, 25].

Many of these expressiveness results relate MSO with automata and modal logics, among which Janin and Walukiewicz’s seminal result [17] showing that the bisimulation-invariant fragment of MSO interpreted over transition systems is captured by the μ -calculus. Notable exceptions to the classical trilogy between MSO , modal logics and automata are MSO on infinite partial orders (see [23] for partial results) and MSO^{eq} (partial results can be found in [25]).

On the other hand, more recently there has been an increased interest in the expressiveness and decidability of logics defined on structures in which two “orthogonal” relations are considered: the so-called temporal epistemic (multi-agent) logics [12], which combine time-passage relations and epistemic relations

Catalin Dima acknowledges financial support from the ANR project EQINOCS.

Bastien Maubert acknowledges financial support from the ERC project EPS 313360.

on the histories of the system [12, 15]. A natural question that arises is whether there exists a natural extension of MSO, of the μ -calculus, and of tree automata for the temporal epistemic framework, and how would they compare?

Note that appropriate extensions of MSO, of the μ -calculus and of tree automata would rely on two sorts of binary relations: those related to dynamic behaviour and those related to epistemic features. While the temporal part of these logics naturally refers to a tree-like structure, the epistemic part requires, in order to model *e.g.* powerful agents with “perfect recall”, to consider binary relations defined on histories. Such models neither are tree-like structures, nor grid-like structures, nor graphs within the Caucal hierarchy. The only proposals in this direction that we know about are [1, 24, 27] and [19]. [27] mentions an encoding of LTL with knowledge into Chain Logic with equal-level predicate, a fragment of MSO^{eq} . [24] introduces the epistemic μ -calculus and studies its model-checking problem. [1] studies temporal logics over tree models with “jump-edges” which capture observational indistinguishability. Each one considers only one particular relation on histories, all three akin to synchronous perfect recall. [19] proposes a generalization of tree automata, called jumping tree automata, applying them to the study of temporal epistemic logics.

In this paper, we propose natural extensions of MSO and the μ -calculus, respectively called *MSO with path relation* ($\text{MSO}^{\curvearrowright}$) and the *jumping μ -calculus* ($\text{L}_{\mu}^{\curvearrowright}$). $\text{MSO}^{\curvearrowright}$ is MSO with an additional binary predicate, \curvearrowright , interpreted “transversely” on the tree structure, according to the path relation. $\text{L}_{\mu}^{\curvearrowright}$ is a generalization of the epistemic μ -calculus defined in [24]: it features a *jumping modality* whose semantics relies on the path relation, generalizing the knowledge operator K . The path relations that we consider are arbitrary and thus cover all variants of indistinguishability relations of, *e.g.* [16], as well as settings not particularly related to any epistemic interpretation [13]. We investigate whether $\text{L}_{\mu}^{\curvearrowright}$ is expressive complete with regards to the bisimulation-invariant fragment of $\text{MSO}^{\curvearrowright}$. For the class of *recognizable* path relations (finite memory), we observe (Theorem 1) that they add no expressivity w.r.t. the classical case hence the expressive completeness holds. We show however that the case of *regular* path relations breaks this completeness: the class of reachability games with imperfect information and *synchronous perfect recall* [5] where the first player wins cannot be defined in the jumping μ -calculus, while being closed under bisimulation and definable in our extension of MSO (Theorem 2). As an intermediate result, we show that the *jumping tree automata* defined in [19] are equivalent to the jumping μ -calculus.

The paper runs as follows: in Sect. 2, we develop the framework of our study on transition systems. We introduce in Sect. 3 the two extensions of MSO and the μ -calculus, and state our results on expressive completeness (Theorems 1 and 2). In Sect. 4 we establish Theorem 1 and prove that jumping automata are equivalent to the jumping μ -calculus, thanks to which we establish succinctness and complexity results on the jumping μ -calculus with recognizable path relation. In Sect. 5 we establish our main result (Theorem 2) by proving that winning in reachability games with imperfect information and synchronous perfect recall

is not definable in the jumping μ -calculus, unless the winning conditions are observable (Theorem 3). We conclude, comment on the impacts of our results, and give some perspectives in Sect. 6.

2 Preliminary Notions

We first fix a few basic notations. Given two words w and w' over some alphabet Σ , we write $w \preceq w'$ if w is a prefix of w' ; if $w = a_0a_1\dots \in \Sigma^\omega$ is an infinite word we let, for each $i \geq 0$, $w[i] := a_i$ and $w[0, i] := a_0a_1\dots a_i$. For a finite word $w = a_0\dots a_{n-1} \in \Sigma^*$, its *length* is $|w| := n$. Also, given a binary relation $R \subseteq A \times B$ for every $a \in A$, we let $R(a) := \{b \mid (a, b) \in R\}$. In the rest of the paper, we fix $\mathcal{AP} = \{p, p', \dots\}$ a countable set of *atomic propositions* and $\mathcal{Act} = \{a, a', \dots\}$ a countable set of *actions*.

Definition 1. A transition system (over \mathcal{AP} and \mathcal{Act}) is a structure $\mathcal{S} = (S, s_\iota, \{a^S\}_{a \in \mathcal{Act}}, \{p^S\}_{p \in \mathcal{AP}})$, where S is a countable set of states, s_ι is an initial state, each a^S is a binary relation over S and each p^S is a subset of S .

The logics that we aim at are concerned with *paths*¹ in transition systems, that is, on their *tree unfoldings*. To ease the presentation, our trees have at most countable branching degree, but, unless otherwise stated, our results still hold for arbitrary degree. A *tree* is a nonempty, prefix-closed set $\tau \subseteq \mathbb{N}^*$. An element $x \in \tau$ is a *node*, and the empty word ϵ is the *root* of the tree. If $x \cdot i \in \tau$, $x \cdot i$ is a *child* of x . A node with no child is a *leaf*. A *branch* is a sequence of nodes in τ (either finite or infinite) in which each node but the first one is a child of the previous one; a branch is *maximal* if it is infinite or it ends up in a leaf. We write $x \preceq y$ if y can be found on some branch that starts in x , and we let $[\tau]_x = \{y \mid x \preceq y\}$ denote the subtree of τ rooted in x . A *markedtree* (over \mathcal{AP} and \mathcal{Act}) is a pair $t = (\tau, m)$, where τ is a tree and $m : \tau \rightarrow (\mathcal{Act} \times 2^{\mathcal{AP}})$ is a *marking* of the nodes, where $m(x) = (a, \ell)$ means that x was reached through action a and ℓ is the set of atomic propositions that hold in x ; we may use notation a^x and ℓ^x for a and ℓ when $m(x) = (a, \ell)$. Node y is an a -child of a node x if y is a child of x and $a^y = a$. The *word* of a node x is $w(x) := m(\epsilon)m(x_1)\dots m(x_n)$, where $\epsilon x_1\dots x_n (= x)$ is the (unique) branch from the root to x . For a finite subset $AP \subset \mathcal{AP}$, an AP -*tree* is a labeled tree $t = (\tau, m)$ such that $\ell^x \subseteq AP$, for every node $x \in t$ (i.e. $x \in \tau$).

Definition 2. Let $\mathcal{S} = (S, s_\iota, \{a^S\}_{a \in \mathcal{Act}}, \{p^S\}_{p \in \mathcal{AP}})$ be a transition system. The unfolding $t_{\mathcal{S}}$ of \mathcal{S} is the markedtree (τ, m) with least tree τ such that: ϵ is associated² to s_ι and $\ell^\epsilon = \{p \mid s_\iota \in p^S\}$, and for each node $x \in \tau$ associated to state s , if $\langle s \xrightarrow{a_i} s_i \rangle_{i \in I}$ is an enumeration of the outgoing transitions from s (with $I \subseteq \mathbb{N}$), then for each $i \in I$ we have $x \cdot i \in \tau$, $x \cdot i$ is associated to s_i and $m(x \cdot i) = (a_i, \{p \in \mathcal{AP} \mid s_i \in p^S\})$.

¹ i.e. finite sequences of states and actions that start in the initial state and follow the binary relations.

² The notion of “associated state” is only used to define unfoldings and is left informal.

Because in the following only actions and atomic propositions matter, the ordering of children nodes in trees is irrelevant, and the unfolding t_S is therefore uniquely defined up to isomorphism.

Definition 3. We call path relation a binary relation over $(Act \times 2^{\mathcal{AP}})^*$.

A path relation links finite paths of transition systems over \mathcal{AP} and Act . It also induces a binary relation between nodes of marked trees (over \mathcal{AP} and Act) in a natural way by relating nodes x and y whenever their words $w(x)$ and $w(y)$ are related. We use notation \rightsquigarrow for path relations.

Finally, we recall the classic notion of *bisimulation* [20].

Definition 4. A bisimulation between transition systems \mathcal{S} and \mathcal{S}' is a binary relation $\mathcal{Z} \subseteq S \times S'$ such that, for all $(s, s') \in \mathcal{Z}$, for all $p \in \mathcal{AP}$ and $a \in Act$:

1. $s \in p^S$ iff $s' \in p^{S'}$;
2. for all $r \in a^S(s)$, there is $r' \in a^{S'}(s')$ such that $(r, r') \in \mathcal{Z}$;
3. and vice-versa.

We write $\mathcal{S} \Leftrightarrow \mathcal{S}'$ whenever there is a bisimulation \mathcal{Z} between \mathcal{S} and \mathcal{S}' such that $(s_i, s'_i) \in \mathcal{Z}$. A class \mathcal{C} of transition systems is *closed under bisimulation*, or *bisimulation closed* if $\mathcal{S} \in \mathcal{C}$ and $\mathcal{S} \Leftrightarrow \mathcal{S}'$ imply $\mathcal{S}' \in \mathcal{C}$, for all \mathcal{S} and \mathcal{S}' .

3 Expressive Completeness Issues

We fix a countable set of *second order variables* $Var = \{X, Y, \dots\}$. Given a marked tree $t = (\tau, m)$, a *valuation* is a mapping $V : Var \rightarrow 2^T$. For $X \in Var$ and $T \subseteq \tau$, we let $V[T/X]$ be the valuation that maps X to T , and which coincides with V on all other variables.

Monadic second order logic with path relation ($\text{MSO}^{\rightsquigarrow}$) is an extension of MSO interpreted over transition systems with a path relation. Its syntax is as follows:

$$\psi ::= sr(X) \mid p(X) \mid succ(X, Y) \mid X \subseteq Y \mid \neg\psi \mid \psi \vee \psi' \mid \exists X.\psi(X) \mid X \rightsquigarrow Y$$

where $p \in \mathcal{AP}$ and $X, Y \in Var$.

An $\text{MSO}^{\rightsquigarrow}$ formula ψ is interpreted over a marked tree $t = (\tau, m)$ with a valuation V and a fixed path relation \rightsquigarrow ; the fact that t with valuation V satisfies ψ is written $t, V \models^{\rightsquigarrow} \psi$, defined inductively as follows:

$$\begin{aligned} t, V \models^{\rightsquigarrow} sr(X) & \text{ if } V(X) = \{\epsilon\} \\ t, V \models^{\rightsquigarrow} p(X) & \text{ if for all } x \in V(X), p \in \ell^x \\ t, V \models^{\rightsquigarrow} succ(X, Y) & \text{ if } V(X) = \{x\}, V(Y) = \{y\}, \text{ and } y \text{ is a child of } x \\ t, V \models^{\rightsquigarrow} X \subseteq Y & \text{ if } V(X) \subseteq V(Y) \\ t, V \models^{\rightsquigarrow} \neg\psi & \text{ if } t, V \not\models^{\rightsquigarrow} \psi \text{ and } t, V \models^{\rightsquigarrow} \psi \vee \psi' \text{ if } t, V \models^{\rightsquigarrow} \psi \text{ or } t, V \models^{\rightsquigarrow} \psi' \\ t, V \models^{\rightsquigarrow} \exists X.\psi(X) & \text{ if there is } T \subseteq t \text{ s.t. } t, V[T/X] \models^{\rightsquigarrow} \psi(X) \\ t, V \models^{\rightsquigarrow} X \rightsquigarrow Y & \text{ if } V(X) = \{x\}, V(Y) = \{y\}, \text{ and } x \rightsquigarrow y \end{aligned}$$

If $\psi \in \text{MSO}^{\rightsquigarrow}$ has no free variable, we simply write $t \models^{\rightsquigarrow} \psi$, and $\mathcal{S} \models^{\rightsquigarrow} \psi$ whenever $t_S \models^{\rightsquigarrow} \psi$, for any transition system \mathcal{S} . Let $\mathcal{L}(\psi, \rightsquigarrow) := \{\mathcal{S} \mid \mathcal{S} \models^{\rightsquigarrow} \psi\}$.

The *jumping μ -calculus*. The syntax of the \curvearrowright -jumping μ -calculus L_μ^{\curvearrowright} is:

$$\varphi ::= X \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \heartsuit\varphi \mid \spadesuit\varphi \mid \mu X.\varphi(X)$$

where $X \in \text{Var}$, $p \in \mathcal{AP}$, $a \in \text{Act}$, and in the last rule, X appears only under an even number of negations (i.e. positively) in $\varphi(X)$. We classically define the dual operators \heartsuit , \spadesuit , \boxplus and ν , e.g. $\boxplus\varphi := \neg\heartsuit\neg\varphi$, and $\nu X.\varphi := \neg\mu X.\neg\varphi$.

Given a path relation \curvearrowright , formulas of L_μ^{\curvearrowright} are interpreted over a marked tree with a valuation $V : \text{Var} \rightarrow 2^\tau$. We inductively define $\llbracket \varphi \rrbracket_{\curvearrowright}^{t,V} \subseteq \tau$ with $t = (\tau, m)$:

$$\begin{aligned} \llbracket X \rrbracket_{\curvearrowright}^{t,V} &= V(X) & \llbracket p \rrbracket_{\curvearrowright}^{t,V} &= \{x \in t \mid p \in \ell^x\} & \llbracket \neg\varphi \rrbracket_{\curvearrowright}^{t,V} &= t \setminus \llbracket \varphi \rrbracket_{\curvearrowright}^{t,V} \\ \llbracket \varphi \vee \varphi' \rrbracket_{\curvearrowright}^{t,V} &= \llbracket \varphi \rrbracket_{\curvearrowright}^{t,V} \cup \llbracket \varphi' \rrbracket_{\curvearrowright}^{t,V} & \llbracket \heartsuit\varphi \rrbracket_{\curvearrowright}^{t,V} &= \{x \in t \mid x \text{ has an } a\text{-child in } \llbracket \varphi \rrbracket_{\curvearrowright}^{t,V}\} \\ \llbracket \spadesuit\varphi \rrbracket_{\curvearrowright}^{t,V} &= \{x \in t \mid \text{there exists } y \in \llbracket \varphi \rrbracket_{\curvearrowright}^{t,V} \text{ such that } x \curvearrowright y\} \\ \llbracket \mu X.\varphi(X) \rrbracket_{\curvearrowright}^{t,V} &= \bigcap \{T \subseteq t \mid \llbracket \varphi(X) \rrbracket_{\curvearrowright}^{t,V[T/X]} \subseteq T\} \end{aligned}$$

Note that, for each formula $\mu X.\varphi(X)$, function $T \mapsto \llbracket \varphi(X) \rrbracket_{\curvearrowright}^{t,V[T/X]}$ is monotone, and hence has a least fixpoint, namely $\llbracket \mu X.\varphi(X) \rrbracket_{\curvearrowright}^{t,V}$. If $\varphi \in L_\mu^{\curvearrowright}$ has no free variables, we write $t \models^{\curvearrowright} \varphi$ whenever $\epsilon \in \llbracket \varphi \rrbracket_{\curvearrowright}^t$, and $\mathcal{S} \models^{\curvearrowright} \varphi$ whenever $t_{\mathcal{S}} \models^{\curvearrowright} \varphi$, for any transition system \mathcal{S} . We let $\mathcal{L}(\varphi, \curvearrowright) := \{\mathcal{S} \mid \mathcal{S} \models^{\curvearrowright} \varphi\}$.

Expressive Completeness. For a logic \mathcal{L} , a class \mathcal{C} of transition systems is \mathcal{L} -*definable* if there is a formula of \mathcal{L} whose set of models is exactly \mathcal{C} .

Proposition 1. *For every path relation \curvearrowright , every L_μ^{\curvearrowright} -definable class is closed under bisimulation.*

This result follows from Lemma 1 below.

Let \curvearrowright be a path relation. Note that a marked tree t can be turned into a transition system t^{\curvearrowright} over \mathcal{AP} and $\text{Act}' := \text{Act} \cup \{\check{a}\}$, where \check{a} is a fresh action symbol, by letting $y \in \check{a}^t(x)$ whenever $x \curvearrowright y$. The following lemma states that if two transition systems are bisimilar (w.r.t. the transition relations only), then their unfoldings enriched with a path relation are bisimilar w.r.t. *both* the transition relations and the path relation.

Lemma 1. *Let \mathcal{S} and \mathcal{S}' be two transition systems, and let \curvearrowright be a path relation. If $\mathcal{S} \cong \mathcal{S}'$, then $t_{\mathcal{S}}^{\curvearrowright} \cong t_{\mathcal{S}'}^{\curvearrowright}$.*

Proposition 2. *Every L_μ^{\curvearrowright} -definable class is $\text{MSO}^{\curvearrowright}$ -definable.*

Proposition 2 can easily be established with a straightforward extension of the effective translation of μ -calculus formulas into MSO given, e.g. in [14, Ch14].

We now engage our main concern, the *expressive completeness* of L_μ^{\curvearrowright} with respect to $\text{MSO}^{\curvearrowright}$. As in [17], due to Proposition 1, this question can only be addressed for bisimulation-closed classes of transition systems. We thus seek properties on the path relation \curvearrowright so that L_μ^{\curvearrowright} is expressive complete with respect to $\text{MSO}^{\curvearrowright}$, in the following sense:

Definition 5 (Expressive completeness). L_μ^\curvearrowright is expressive complete with respect to MSO^\curvearrowright if every bisimulation-closed class of transition systems that is MSO^\curvearrowright -definable is also L_μ^\curvearrowright -definable.

Note that the results in [17] do not adapt to MSO^\curvearrowright since unfoldings of transition systems with a path relation are not tree-like structures.

We briefly recall the notions of recognizable, regular and rational relations, and we refer to [3] for details. Let Σ be a finite alphabet. A binary relation over Σ^* is *rational* if there is a *transducer* (i.e. a finite-state two-tape automaton) that accepts precisely the pairs of words of the relation. A binary relation (over Σ^*) is *regular* if it is accepted by a *synchronous transducer*³. The epistemic relation of an agent with asynchronous perfect recall [22] is rational whereas the epistemic relation of an agent with synchronous perfect recall [4, 5] is regular. A binary relation is *recognizable* if there is a finite-state word automaton over $\Sigma \cup \{\#\}$ that accepts words of the form $w\#w'$, whenever w and w' are related. The epistemic relation of a “bounded-memory” agent is recognizable. We say that a path relation \curvearrowright (over $(Act \times 2^{AP})^*$) is *rational* (resp. *regular*, *recognizable*) if there are finite subsets $A \subset Act$ and $AP \subset \mathcal{AP}$ such that \curvearrowright is equal to some rational (resp. regular, recognizable) relation over $(A \times 2^{AP})^*$. Finally, recall that recognizable relations are strictly contained in regular relations, and so are regular relations in rational relations.

Theorem 1. For any recognizable path relation, L_μ^\curvearrowright is expressive complete with respect to MSO^\curvearrowright .

Theorem 2. There are regular (hence rational) binary relations for which L_μ^\curvearrowright is not expressive complete with respect to MSO^\curvearrowright .

4 Tree Automata for the jumping μ -calculus

We prove that *jumping tree automata* (JTA), introduced in [19], are equivalent to the jumping μ -calculus, which entails complexity and succinctness results for L_μ^\curvearrowright with recognizable path relation, and we prove Theorem 1. We assume familiarity with two-player turn-based games.

For a set X , $\mathbb{B}^+(X)$ (with typical elements $\alpha, \beta \dots$) is the set of formulas built with elements of X as atomic propositions using only connectives \vee and \wedge , and with $\top \in \mathbb{B}^+(X)$. Let $Dir = \{\diamond \mid a \in Act\} \cup \{\boxminus \mid a \in Act\} \cup \{\heartsuit, \boxplus\}$ be the set of *automaton directions*.

Definition 6. A jumping tree automaton (JTA) over AP is a structure $\mathcal{A} = (AP, Q, q_i, \delta, C)$ where $AP \subset \mathcal{AP}$ is a finite set of atomic propositions, Q is a finite set of states, $q_i \in Q$ is an initial state, $\delta : Q \times 2^{AP} \rightarrow \mathbb{B}^+(Dir \times Q)$ is a transition function, and $C : Q \rightarrow \mathbb{N}$ is a colouring function.

³ i.e. it progresses at the same pace on each tape.

JTAs resemble alternating tree automata [14, Ch.9]. *Action directions* (i.e. \blacklozenge and \blackbox) are meant to go down the input tree, whereas the new *jump directions* \blacklozenge and \blackbox rely on an *a priori* given path relation. A JTA \mathcal{A} with a path relation \rightsquigarrow is written $(\mathcal{A}, \rightsquigarrow)$. Acceptance is defined on a two-player parity game between Eve and Adam. Let $t = (\tau, m)$ be an AP-tree, and let $\mathcal{A} = (AP, Q, q_i, \delta, C)$. We define the parity game $\mathcal{G}_{\mathcal{A}, \rightsquigarrow}^t = (V, v_i, E, V_E, V_A, C')$ whose set of positions is $V = \tau \times Q \times \mathbb{B}^+(Dir \times Q)$, whose initial position is $v_i = (\epsilon, q_i, \delta(q_i, \ell^\epsilon))$, and where a position (x, q, α) belongs to Eve iff α is of the form $\alpha_1 \vee \alpha_2$, $[\blacklozenge, q']$, or $[\blackbox, q']$. The possible moves in $\mathcal{G}_{\mathcal{A}, \rightsquigarrow}^t$ are the following:

$$\begin{aligned} (x, q, \alpha_1 \dagger \alpha_2) &\rightarrow (x, q, \alpha_i) && \text{where } \dagger \in \{\vee, \wedge\} \text{ and } i \in \{1, 2\} \\ (x, q, [\blacklozenge, q']) &\rightarrow (y, q', \delta(q', \ell^y)) && \text{where } \blacklozenge \in \{\blacklozenge, \blackbox\} \text{ and } y \text{ is an } a\text{-child of } x \\ (x, q, [\blackbox, q']) &\rightarrow (y, q', \delta(q', \ell^y)) && \text{where } \blackbox \in \{\blacklozenge, \blackbox\} \text{ and } x \rightsquigarrow y \end{aligned}$$

Deadlock positions are winning for Eve if of the form (x, q, \top) , $(x, q, [\blackbox, q'])$ or $(x, q, [\blacklozenge, q'])$, and winning for Adam otherwise. The colouring function C' of $\mathcal{G}_{\mathcal{A}, \rightsquigarrow}^t$ is inherited from the one of \mathcal{A} by letting $C'(x, q, \alpha) = C(q)$. We let $\mathcal{L}(\mathcal{A}, \rightsquigarrow) = \{\mathcal{S} \mid \text{Eve has a winning strategy in } \mathcal{G}_{\mathcal{A}, \rightsquigarrow}^{\mathcal{S}}\}$.

Proposition 3.

- (a) For every formula $\varphi \in \mathbb{L}_\mu^{\rightsquigarrow}$, there is a JTA \mathcal{A}_φ such that, for every path relation \rightsquigarrow , $\mathcal{L}(\varphi, \rightsquigarrow) = \mathcal{L}(\mathcal{A}_\varphi, \rightsquigarrow)$,
- (b) for every JTA \mathcal{A} , there is an $\mathbb{L}_\mu^{\rightsquigarrow}$ -formula $\varphi_{\mathcal{A}}$ such that, for every path relation \rightsquigarrow , $\mathcal{L}(\mathcal{A}, \rightsquigarrow) = \mathcal{L}(\varphi_{\mathcal{A}}, \rightsquigarrow)$.

Moreover, the translations are effective and linear.

When restricting to recognizable path relations, the folklore fact that recognizable relations are MSO-definable gives the following.

Proposition 4. *MSO $^{\rightsquigarrow}$ with recognizable path relations is not more expressive than MSO.*

Theorem 1 is obtained from Propositions 2 and 4 and the expressive completeness of the μ -calculus w.r.t. MSO. This collapse of the jumping μ -calculus down to the μ -calculus uses transformations that do not provide accurate complexity bounds regarding the jumping μ -calculus. However, by Proposition 3 and the relationship between two-way alternating automata and classic tree automata [28] we get:

Proposition 5. *The satisfiability problem for the jumping μ -calculus with recognizable path relations over transition systems with bounded branching degree is EXPTIME-complete.*

Proposition 6. *For a fixed recognizable path relation, the jumping μ -calculus with path relations over transition systems with bounded branching degree is at most exponentially more succinct than the μ -calculus.*

5 Games and the jumping μ -calculus

We focus on the property stating the existence of a winning strategy in two-player turned-based reachability games with imperfect information. This property gives us Theorem 2, for being bisimilar-invariant, expressible in $\text{MSO}^{\curvearrowright}$, but not expressible in $\text{L}_{\mu}^{\curvearrowright}$ (Theorem 4).

Two-player Games with Imperfect Information [2,9]. The players are Eveⁱ and Adamⁱ. Eveⁱ partially observes the positions, while Adamⁱ has perfect information. Let $\mathcal{Obs} = \{o, o', \dots\}$ be a countable set of *observations*. An *imperfect-information game arena* is a tuple $G^i = (V, v_l, \{a^{G^i}\}_{a \in \mathcal{Act}}, \{o^{G^i}\}_{o \in \mathcal{Obs}})$, where V is a set of positions, $v_l \in V$ is an initial position, each a^{G^i} is a binary relation over V and each o^{G^i} is a subset of V such that $\{o^{G^i}\}_{o \in \mathcal{Obs}}$ forms a partition of V . An action $a \in \mathcal{Act}$ is *available* in $v \in V$ if $a^{G^i}(v) \neq \emptyset$. We assume that some action is available in every position, and that two positions with the same observation share the same available actions. For $v \in V$, write o_v the unique observation such that $v \in o_v$.

Players take turns, starting with Eveⁱ. In current position v , Eveⁱ chooses an available action a and Adamⁱ chooses a new position $v' \in a^{G^i}(v)$. A *play* (resp. *partial play*) is an infinite (resp. finite) sequence $\pi = v_0 a_1 v_1 a_2 \dots$ (resp. $\rho = v_0 a_1 v_1 \dots a_n v_n$) such that $v_0 = v_l$ and, for all i , $v_{i+1} \in a_{i+1}^{G^i}(v_i)$. In the *synchronous perfect recall* setting [5], Eveⁱ remembers the whole sequence of observations that she receives, and her actions. The *indistinguishability equivalence* over partial plays is thus defined by: for $\rho = v_0 a_1 \dots a_n v_n$ and $\rho' = v'_0 a'_1 \dots a'_n v'_n$, we let $\rho \sim \rho'$ whenever, for all $1 \leq i \leq n$, $o_{v_i} = o_{v'_i}$ and $a_i = a'_i$.

Remark 1. Here, Eve remembers the sequence of actions. We point out that if she does not, then Theorem 3 below does not hold.

A (uniform) *strategy* for Eve is a partial function $\sigma : \{v_l\}(\mathcal{Act} \cdot V)^* \rightarrow \mathcal{Act}$ such that for two partial plays ρ and ρ' , if $\rho \sim \rho'$, then $\sigma(\rho) = \sigma(\rho')$. We say that a play $\pi = v_0 a_1 v_1 \dots$ *follows* a strategy σ if for all $i \geq 0$, $a_{i+1} = \sigma(v_0 a_1 v_1 \dots a_i v_i)$.

A (*reachability*) *game with imperfect information* \mathcal{G}^i is an imperfect-information arena $G^i = (V, v_l, \{a^{G^i}\}_{a \in \mathcal{Act}}, \{o^{G^i}\}_{o \in \mathcal{Obs}})$ together with a reachability winning condition $F \subseteq V$. A strategy for Eveⁱ is *winning* if every play that follows it visits F . F is *observable* if for all positions v and v' , $o_v = o_{v'}$ implies $(v \in F \Leftrightarrow v' \in F)$.

Note that a game with imperfect information is a transition system over $\mathcal{AP} = \mathcal{Obs} \cup \{F\}$ and \mathcal{Act} . Note also that the relation \sim on partial plays induces a path relation, that we shall also write \sim .

Consider the class $\mathcal{R}(A, O)$ of reachability games with imperfect information where actions range over $A \subseteq \mathcal{Act}$ and observations range over $O \subseteq \mathcal{Obs}$. To address logical definability, we restrict to games where A and O are finite sets. Also, because we address μ -calculus definability, we close by bisimulation, so that from now on a *game* is a transition system whose subsystem connected to the initial position is a game as defined above.

Notice that the subclass $\mathcal{R}(A, O)$ where Eveⁱ wins⁴ is bisimulation closed since, according to [5], for any two imperfect-information reachability games \mathcal{G}^i and $\mathcal{G}^{i'}$, if $\mathcal{G}^i \Leftrightarrow \mathcal{G}^{i'}$, then Eveⁱ wins in \mathcal{G}^i if, and only if, she wins in $\mathcal{G}^{i'}$.

Theorem 3. *The subclass of $\mathcal{R}(A, O)$ of games with observable winning condition where Eveⁱ wins is definable in L_μ^\sim , namely by $\mu X.(F \vee \bigvee_{a \in A} \Box \Box X)$.*

Actually, a result similar to Theorem 3 can be established for parity conditions.

Theorem 4. *The subclass of $\mathcal{R}(A, O)$ where Eveⁱ wins is not L_μ^\sim -definable.*

Theorem 4 entails Theorem 2 since the subclass of $\mathcal{R}(A, O)$ where Eveⁱ wins is clearly $\text{MSO}^{\sphericalangle}[\sim]$ -definable, closed under bisimulation, and the synchronous perfect-recall relation \sim is regular⁵.

Proof of Theorem 4. We prove that Theorem 4 holds, already when we consider only two actions and one observation, *i.e.* $A = \{a_0, a_1\}$ and $O = \{o\}$.

The proof is dealt with by contradiction: Assume that there is a formula $\Phi_{\text{Win}} \in L_\mu^\sim$ such that for every $\mathcal{G}^i \in \mathcal{R}(A, O)$, $\mathcal{G}^i \models \Phi_{\text{Win}}$ if, and only if, Eveⁱ has a winning strategy in \mathcal{G}^i . By Proposition 3, there is a JTA $\mathcal{A} = (AP, Q, q_\iota, \delta, C)$ such that $\mathcal{L}(\Phi_{\text{Win}}, \sim) = \mathcal{L}(\mathcal{A}, \sim)$. Let $N := |Q| + 1$.

We describe 2^N (unfoldings of) games in $\mathcal{R}(A, O)$ where Eveⁱ wins. For each one, we exhibit a winning strategy in the (perfect information) acceptance game of \mathcal{A} on this unfolding. We then employ the ‘‘pigeon hole’’ principle to show that at least two of these ‘‘accepting’’ strategies can be combined into a new ‘‘accepting’’ strategy of Eve, entailing acceptance by \mathcal{A} of a game in $\mathcal{R}(A, O)$ where Eveⁱ has no winning strategy, hence the contradiction.

The family of game unfoldings that we consider is depicted in Fig. 1. They all share the same unmarked tree τ , and for all $k \in \{1, \dots, 2^N + 2\}$, $[\tau]_{y_k}$ is the full binary tree with action a_0 (resp. a_1) leading to the left (resp. right) child. Let $w_k \in \{0, 1\}^N$ be the binary representation of $k - 1$. Between the different trees t_i , the markings only differ on the leaves of $[\tau]_{y_{2^N+1}}$ and $[\tau]_{y_{2^N+2}}$: for $1 \leq i \leq 2^N$, in $[t_i]_{y_{2^N+1}}$ and $[t_i]_{y_{2^N+2}}$, the only nodes in F are $y_{2^N+1} \cdot w_i$ and $y_{2^N+2} \cdot w_i$. Finally, since Eveⁱ is blind, her strategies are simply described by sequences of actions.

For each $1 \leq i \leq 2^N$, write $\mathcal{G}_i = (V^i, v_i^i, E^i, V_E^i, V_A^i, C^i)$ for $\mathcal{G}_{\mathcal{A}, \sim}^{t_i}$, *i.e.* the acceptance game of \mathcal{A} on t_i with relation \sim .

Clearly, for each $1 \leq i \leq 2^N$, Eveⁱ wins t_i , and thus Eve wins \mathcal{G}_i . Let σ_i be a winning strategy for Eve in each game \mathcal{G}_i . Let $\text{visit}_{\sigma_i} : \tau \rightarrow 2^Q$ associate to each node of τ the set of states of \mathcal{A} visited by strategy σ_i : formally, $\text{visit}_{\sigma_i}(x) := \{q \mid \exists \pi \in \text{Out}(\mathcal{G}_i, \sigma_i), \exists n \geq 0, \exists \alpha \in \mathcal{B}^+(\text{Dir} \times Q) \text{ s.t. } \pi[n] = (x, q, \alpha)\}$.

Since there are at most $2^{|Q|}$ different such sets of states, and we have 2^N strategies with $N = |Q| + 1$, there exist $i \neq j$ s.t. $\text{visit}_{\sigma_i}(y_{2^N+1}) = \text{visit}_{\sigma_j}(y_{2^N+1})$. Fix such a pair (i, j) , and define the game unfolding t_0 , obtained from t_i by

⁴ *i.e.* has a winning strategy.

⁵ a one-state transducer that accepts it can easily be exhibited.

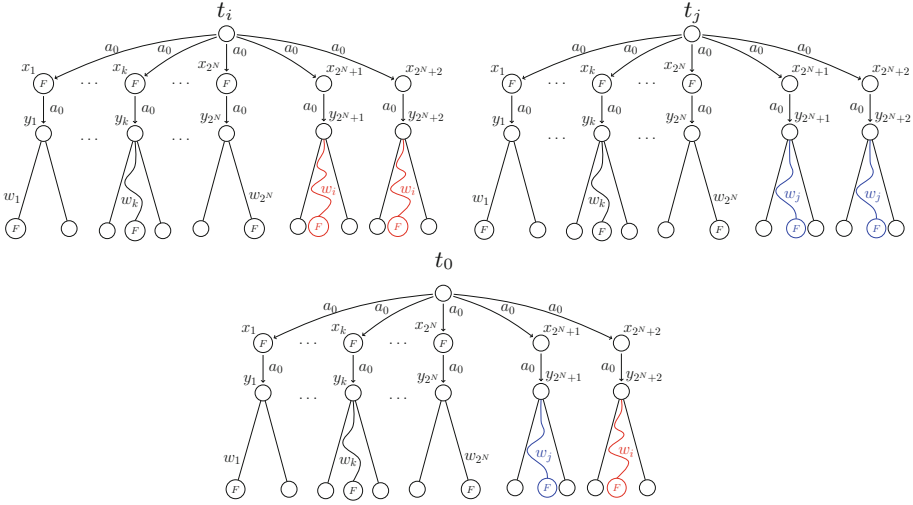


Fig. 1. The tree t_i , the tree t_j , and the hybrid tree t_0 .

replacing the subtree $[t_i]_{y_{2^N+1}}$ with $[t_j]_{y_{2^N+1}}$ (see Fig. 1). Note that t_0 is the unfolding of a game in $\mathcal{R}(A, O)$. Let us write $\mathcal{G}_0 = (V^0, v_\ell^0, E^0, V_E^0, V_A^0, C^0)$ for $\mathcal{G}_{\mathcal{A}, \sim}^{t_0}$. Observe that the three games \mathcal{G}_i , \mathcal{G}_j and \mathcal{G}_0 share the same set of positions: $V^0 = V^i = V^j = \tau \times Q \times \mathbb{B}^+(Dir \times Q)$ that we now write V . Also, for all $1 \leq k \leq 2^N + 2$, $\ell_0^{y_k} = \ell_i^{y_k} = \ell_j^{y_k} (= \{o\})$, that we now write ℓ . We succinctly write v_k^q for positions $(y_k, q, \delta(q, \ell))$, which play an important role.

The following lemma allows us to transfer the existence of winning strategies in positions v_k^q from \mathcal{G}_i and \mathcal{G}_j to \mathcal{G}_0 .

Lemma 2. 1. For all $q \in Q$ and $k \neq 2^N + 1$, $(\mathcal{G}_0, v_k^q) \Leftrightarrow (\mathcal{G}_i, v_k^q)$, and
 2. for all $q \in Q$ and $k \neq 2^N + 2$, $(\mathcal{G}_0, v_k^q) \Leftrightarrow (\mathcal{G}_j, v_k^q)$.

By design of t_0 , Eveⁱ has no winning strategy in t_0 . Therefore, $t_0 \not\approx \Phi_{Win}$, and thence $t_0 \notin \mathcal{L}(\mathcal{A}, \sim)$, i.e. Eve does not have a winning strategy in the acceptance game \mathcal{G}_0 of JTA \mathcal{A} on t_0 with path relation \sim . We establish Proposition 7 below, which provides a contradiction and terminates the proof of Theorem 4.

Proposition 7. *Eve has a winning strategy in \mathcal{G}_0 .*

Proof sketch. Let us define $Start_\tau := \{\epsilon, x_1, \dots, x_{2^N+2}\}$, the two first levels of τ , and $Start_{\mathcal{G}} := \{(x, q, \alpha) \in V \mid x \in Start_\tau\}$. Every play in \mathcal{G}_0 starts in $Start_{\mathcal{G}}$.

Note that from any position of $Start_{\mathcal{G}}$, the same moves are available in \mathcal{G}_0 , \mathcal{G}_i and \mathcal{G}_j . In \mathcal{G}_0 , we let Eve follow σ_i as long as the game is in $Start_{\mathcal{G}}$. If the game remains in $Start_{\mathcal{G}}$ for ever (by jumping infinitely), the obtained play is an outcome of σ_i , which is winning for Eve in \mathcal{G}_i . Because positions have the same colour in all acceptance games, this play is also winning for Eve in \mathcal{G}_0 . Otherwise, the play exits $Start_{\mathcal{G}}$ by going down the tree, hence it reaches some position v_k^q .

Because v_k^q has been reached by the winning strategy σ_i , it is a winning position for Eve in the perfect-information parity game \mathcal{G}_i . If $k \neq 2^N + 1$, by Point 1 of Lemma 2, *i.e.* $(\mathcal{G}_0, v_k^q) \Leftrightarrow (\mathcal{G}_i, v_k^q)$, Eve also has a winning strategy from v_k^q in \mathcal{G}_0 . If $k = 2^N + 1$, because $\text{visit}_{\sigma_i}(y_{2^N+1}) = \text{visit}_{\sigma_j}(y_{2^N+1})$, σ_j also visits position $v_k^q[2^N + 1]$, and therefore $v_k^q[2^N + 1]$ is a winning position for Eve in \mathcal{G}_j . By Point 2 of Lemma 2, *i.e.* $\mathcal{G}_0, v_k^q \Leftrightarrow \mathcal{G}_j, v_k^q$, Eve also has a winning strategy from v_k^q in \mathcal{G}_0 . \square

6 Conclusion and Perspectives

We have considered a general setting where transition systems are equipped with path relations. We have proposed natural extensions of MSO and the μ -calculus in this setting: $\text{MSO}^\curvearrowright$, which is MSO with path relations, and $\text{L}_\mu^\curvearrowright$, which is the jumping μ -calculus. We have studied the question of whether the bisimulation-invariant fragment of $\text{MSO}^\curvearrowright$ and $\text{L}_\mu^\curvearrowright$ have the same expressivity, like in [17]. In the case of recognizable relations, the whole picture collapses to the classic case (Theorem 1). However, for the synchronous perfect recall path relation (a regular binary relation that captures models of agency with time and knowledge), the answer is negative (Theorem 2).

Our results suggest that the adequate logic on transition systems with path relations may lie in between $\text{L}_\mu^\curvearrowright$ and $\text{MSO}^\curvearrowright$: on the one hand, the latter is undecidable for regular path relations as simple as the “Equal Level” relation [25]; on the other hand fundamental decidable properties, such as winning in two-player imperfect-information games with perfect recall, are not captured by the former (Theorem 4). Remark that this property is expressible in Alternating-time Temporal Logic with imperfect information, which is therefore not subsumed by the jumping mu-calculus, but whose model-checking is decidable for one agent with synchronous perfect recall.

In addition, both the epistemic mu-calculus and imperfect-information games are decidable for perfect-recall when the knowledge of the agents is hierarchically ordered [7, 21]. Recent results on games with imperfect information also signal other classes of models with several agents and perfect recall that can be handled [6]. So even for the case of several relations on paths, a logic that would encompass both the (jumping) mu-calculus and games with imperfect information may have good computational properties on interesting classes of systems.

References

1. Alur, R., Černý, P., Chaudhuri, S.: Model checking on trees with path equivalences. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 664–678. Springer, Heidelberg (2007)
2. Apt, K.R., Grädel, E.: Lectures in Game Theory for Computer Scientists. Cambridge University Press, Cambridge (2011)
3. Berstel, J.: Transductions and Context-free Languages, vol. 4. Teubner, Stuttgart (1979)

4. Berwanger, D., Chatterjee, K., De Wulf, M., Doyen, L., Henzinger, T.A.: Strategy construction for parity games with imperfect information. *Inf. Comput.* **208**(10), 1206–1220 (2010)
5. Berwanger, D., Kaiser, L.: Information tracking in games on graphs. *J. Logic Lang. Inf.* **19**(4), 395–412 (2010)
6. Berwanger, D., Mathew, A.B.: Games with recurring certainty. In: *Proceedings of SR Games with Recurring Certainty 2014*, pp. 91–96 (2014)
7. Bozianu, R., Dima, C., Enea, C.: Model checking an epistemic mu-calculus with synchronous and perfect recall semantics. In: *TARK 2013* (2013)
8. Caucal, D.: On infinite transition graphs having a decidable monadic theory. *Theor. Comput. Sci.* **290**(1), 79–115 (2003)
9. Chatterjee, K., Doyen, L.: The complexity of partial-observation parity games. In: Fermüller, C.G., Voronkov, A. (eds.) *LPAR-17. LNCS*, vol. 6397, pp. 1–14. Springer, Heidelberg (2010)
10. Courcelle, B., Engelfriet, J.: *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*. Cambridge University Press, Cambridge (2012)
11. Elgot, C.C., Rabin, M.O.: Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. *J. Symb. Log.* **31**(2), 169–181 (1966)
12. Fagin, R., Halpern, J., Moses, Y., Vardi, M.: *Reasoning about knowledge*. The MIT Press, Boston (2004)
13. Grädel, E.: Model-checking games for logics of imperfect information. *Theor. Comput. Sci.* **493**, 2–14 (2013)
14. Farwer, B.: 1 omega-Automata. In: Grädel, E., Thomas, W., Wilke, T. (eds.) *Automata, Logics, and Infinite Games. LNCS*, vol. 2500, pp. 3–21. Springer, Heidelberg (2002)
15. Halpern, J.Y., van der Meyden, R., Vardi, M.Y.: Complete axiomatizations for reasoning about knowledge and time. *SIAM J. Comput.* **33**(3), 674–703 (2004)
16. Halpern, J.Y., Vardi, M.Y.: The complexity of reasoning about knowledge and time. 1. Lower bounds. *J. Comp. Sys. Sci.* **38**(1), 195–237 (1989)
17. Janin, D., Walukiewicz, I.: On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In: Sassone, V., Montanari, U. (eds.) *CONCUR 1996. LNCS*, vol. 1119. Springer, Heidelberg (1996)
18. Matz, O., Schweikardt, N., Thomas, W.: The monadic quantifier alternation hierarchy over grids and graphs. *Inf. Comput.* **179**(2), 356–383 (2002)
19. Maubert, B., Pinchinat, S.: Jumping automata for uniform strategies. In: *Proceedings of FSTTCS 2013*, pp. 287–298 (2013)
20. Park, D.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) *Proceedings of 5th GI Conference on Theoretical Computer Science. Lecture Notes in Computer Science*, vol. 104, pp. 167–183. Springer, Heidelberg (1981)
21. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: *POPL 1989*, pp. 179–190. ACM Press (1989)
22. Puchala, B.: Asynchronous omega-regular games with partial information. In: Hliněný, P., Kučera, A. (eds.) *MFCS 2010. LNCS*, vol. 6281, pp. 592–603. Springer, Heidelberg (2010)
23. Reiter, F.: Distributed graph automata. CoRR: [abs/1404.6503](https://arxiv.org/abs/1404.6503) (2014)
24. Shilov, N.V., Garanina, N.O.: Combining knowledge and fixpoints. Technical report Preprint n.98, A.P. Ershov Institute of Informatics Systems, Novosibirsk 2002. <http://www.iis.nsk.su/files/preprints/098.pdf>
25. Thomas, W.: Infinite trees and automaton-definable relations over omega-words. *Theor. Comput. Sci.* **103**(1), 143–159 (1992)

26. Thomas, W.: Languages, automata, and logic. In: Rozenberg, G., Salomaa, A. (eds.) *Handbook of Formal Languages: Volume 3 Beyond Words*, pp. 389–455. Springer, Heidelberg (1997)
27. van der Meyden, R., Shilov, N.V.: Model checking knowledge and time in systems with perfect recall. In: Pandu Rangan, C., Raman, V., Sarukkai, S. (eds.) *FST TCS 1999*. LNCS, vol. 1738, pp. 432–445. Springer, Heidelberg (1999)
28. Vardi, M.Y.: Reasoning about the past with two-way automata. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) *ICALP 1998*. LNCS, vol. 1443, pp. 628–641. Springer, Heidelberg (1998)
29. Walukiewicz, I.: Monadic second-order logic on tree-like structures. *Theor. Comput. Sci.* **275**(1–2), 311–346 (2002)

Weighted Automata and Logics on Graphs

Manfred Droste and Stefan Dück^(✉)

Institute of Computer Science, Leipzig University, D-04109 Leipzig, Germany
{droste, dueck}@informatik.uni-leipzig.de

Abstract. Weighted automata model quantitative features of the behavior of systems and have been investigated for various structures like words, trees, traces, pictures, and nested words. In this paper, we introduce a general model of weighted automata acting on graphs, which form a quantitative version of Thomas' unweighted model of graph acceptors. We derive a Nivat theorem for weighted graph automata which shows that their behaviors are precisely those obtainable from very particular weighted graph automata and unweighted graph acceptors with a few simple operations. We also show that a suitable weighted MSO logic is expressively equivalent to weighted graph automata. As a consequence, we obtain corresponding Büchi-type equivalence results known from the recent literature for weighted automata and weighted logics on words, trees, pictures, and nested words. Establishing such a general result has been an open problem for weighted logic for some time.

Keywords: Quantitative automata · Graphs · Quantitative logic · Weighted automata · Büchi · Nivat

1 Introduction

In automata theory, the fundamental Büchi-Elgot-Trakhtenbrot theorem [6, 16, 35] established the coincidence of regular languages with languages definable in monadic second order logic. This led both to practical applications, e.g. in verification of finite-state programs, and to important extensions covering, for example, trees [7, 28, 31], traces [32], pictures [18], nested words [1], and texts [21]; a general result for graphs was given in [33].

At the same time as Büchi, Schützenberger [30] introduced weighted finite automata and characterized their quantitative behaviors as rational formal power series. This model also quickly developed a flourishing theory (see [3, 11, 15, 22, 29]). Recently, Droste and Gastin [9] introduced a weighted MSO logic and showed its expressive equivalence to weighted automata. Soon, different authors extended this result to weighted automata and weighted logics on trees [14], traces [25], pictures [17], nested words [8, 24], and texts [23].

However, a general result for weighted automata and weighted logics covering graphs and linking the previous results remained, up to now, open. The main contributions of this paper are the following.

S. Dück—supported by Deutsche Forschungsgemeinschaft (DFG), project DR 202/11-1 and Graduiertenkolleg 1763 (QuantLA).

- We establish a model of weighted automata on graphs, which extends both Thomas' graph acceptors [34] and the previous weighted automata models for words, trees, pictures, and others. We show that this enables us to model new quantitative properties of graphs which could not be expressed by the previous models.
- To show the robustness of our model, we extend a classical result of Nivat [27] to weighted automata on graphs, showing that their behaviors are exactly those which can be constructed from very particular weighted graph automata and recognizable graph languages using operations like morphisms and intersections.
- We derive a Büchi-type equivalence result for the expressive power of weighted automata and a suitable weighted logic on graphs. We obtain corresponding equivalence results for structures like words, trees, pictures, and nested words as a consequence.
- We show that if the underlying semiring of weights is idempotent, then both our Nivat and Büchi type equivalence results can be sharpened, but not in general.

We note that an interesting approach connecting pebble navigating weighted automata and weighted first-order logic was given in [5, 26]. The present automata model is different, using tiles of graphs.

In our proofs, in comparison to the situation for words, trees, or pictures, several difficulties arise. The crucial difference to previous approaches is a global acceptance condition in form of a check of occurrence numbers of finite tiles in runs of our automata on graphs; we need this condition to connect logic and automata. Furthermore, since we are dealing with graphs, the underlying unweighted automata model cannot be determinized. Accordingly, the unweighted logic subfragment covers only existential MSO. Also, the closure under weighted universal quantifications requires new methods; here we employ a theorem of Thomas [34] whose proof in turn relied on Hanf's theorem [20]. Finally, in contrast to words, trees, pictures, or nested words, our general graphs do not have a distinguished vertex, which yields technical complications.

All our constructions of weighted graph automata are effective. For technical simplicity, here we assume that the weights stem from a commutative semiring. Several constructions would also work for general semirings or for e.g. average computations of weights; this will be later work. It is also tempting to investigate now how the rich field of weighted graph algorithms could be utilized, e.g., for developing decision procedures for weighted graph automata.

2 Graph Acceptors

In this section, we introduce the basic concepts around graphs and graph acceptors. Following [34], we define a (*directed*) *graph* as a relational structure $G = (V, (P_a)_{a \in A}, (E_b)_{b \in B})$ over two finite alphabets A and B , where V is the set of *vertices*, the sets P_a ($a \in A$) form a partition of V , and the sets E_b

($b \in B$) are disjoint irreflexive binary relations on V , called *edges*. We denote with $E = \bigcup_{b \in B} E_b$ the set of all edges. A graph is *bounded by t* if every vertex has an (in- plus out-) degree smaller than or equal to t . We denote with $\text{DG}_t(A, B)$ the class of all finite directed graphs over A and B bounded by t .

We call a class of graphs *pointed* if every graph G of this class is *pointed with a vertex v* , i.e. it has a unique designated vertex $v \in V$. Formally, this assumption can be defined by adding a unary relation *center* to G with center = $\{v\}$.

Let $r \geq 0$. We say the distance of u and v is at most r if there exists a path ($u = u_0, u_1, \dots, u_j = v$) with $j \leq r$ and $(u_i, u_{i+1}) \in E$ or $(u_{i+1}, u_i) \in E$ for all $i < j$. We denote with $\text{sph}^r(G, v)$, the *r -sphere of G around the vertex v* , the unique subgraph of G pointed with v and consisting of all vertices with a distance to v of at most r , together with their edges. We call $\tau = (G, v)$ an *r -tile* if $(G, v) = \text{sph}^r(G, v)$, and may omit the r if the context is clear. The bound t ensures that there exists only finitely many pairwise non-isomorphic r -tiles.

Definition 1 ([33,34]). A graph acceptor (GA) \mathcal{A} over the alphabets A and B is defined as a quadruple $\mathcal{A} = (Q, \Delta, \text{Occ}, r)$ where

- Q is a finite set of states,
- $r \in \mathbb{N}_0$, the tile-size,
- Δ is a finite set of pairwise non-isomorphic r -tiles over $A \times Q$ and B ,
- Occ , the occurrence constraint, is a boolean combination of formulas “ $\text{occ}(\tau) \geq n$ ”, where $n \in \mathbb{N}$ and $\tau \in \Delta$

Given a graph G of $\text{DG}_t(A, B)$ and a mapping $\rho : V \rightarrow Q$, we consider the Q -labeled graph $G_\rho \in \text{DG}_t(A \times Q, B)$, obtained by labeling every vertex $v \in V$ also with $\rho(v)$. We call ρ a *run (or tiling) of \mathcal{A} on G* if for every $v \in V$, $\text{sph}^r(G_\rho, v)$ is isomorphic to a tile in Δ .

We say G_ρ *satisfies* $\text{occ}(\tau) \geq n$ if there exist at least n vertices $v \in V$ such that $\text{sph}^r(G_\rho, v)$ is isomorphic to τ . The semantics of G_ρ *satisfies* Occ is then defined in the usual way. We call a run ρ *accepting* if G_ρ satisfies the constraint Occ . We let $L(\mathcal{A}) = \{G \in \text{DG}_t(A, B) \mid \text{there exists an accepting run } \rho : V \rightarrow Q \text{ of } \mathcal{A} \text{ on } G\}$, the *language accepted by \mathcal{A}* . We call a language $L \subseteq \text{DG}_t(A, B)$ *recognizable* if $L = L(\mathcal{A})$ for some GA \mathcal{A} .

Next, we introduce the logic $\text{MSO}(\text{DG}_t(A, B))$, short MSO , cf. [34]. We denote with x, y, \dots and X, Y, \dots first- and second-order variables ranging over vertices, resp. over sets of vertices. The formulas of MSO are defined inductively by

$$\varphi ::= P_a(x) \mid E_b(x, y) \mid x = y \mid x \in X \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

where $a \in A$ and $b \in B$. An *FO-formula* is a formula of MSO without $\exists X$. An *EMSO-formula* is a formula of the form $\exists X_1 \dots \exists X_k.\varphi$ where φ is an FO-formula.

The satisfaction relation \models for graphs and MSO -sentences is defined in the natural way. For a sentence $\varphi \in \text{MSO}$, we define the *language of φ* as $L(\varphi) = \{G \in \text{DG}_t(A, B) \mid G \models \varphi\}$. We call a language $L \subseteq \text{DG}_t(A, B)$ *MSO-* (resp. *FO-*) *definable* if $L = L(\varphi)$ for some MSO - (resp. FO -) sentence φ .

Theorem 2 ([34]). *Let $L \subseteq \text{DG}_t(A, B)$ be a set of graphs. Then:*

1. L is recognizable by a one-state GA iff L is definable by an FO-sentence.
2. L is recognizable iff L is definable by an EMSO-sentence.

3 Weighted Graph Automata

In this section, we introduce and investigate a quantitative version of graph acceptors. In the following, let $\mathbb{K} = (K, +, \cdot, 0, 1)$ be a commutative semiring, i.e. $(K, +, 0)$ and $(K, \cdot, 1)$ are commutative monoids, $(x+y) \cdot z = x \cdot z + y \cdot z$ and $0 \cdot x = x \cdot 0 = 0$ for all $x, y, z \in K$. Important examples of commutative semirings are the Boolean semiring $\mathbb{B} = (\{0, 1\}, \vee, \wedge, 0, 1)$, the semiring of the natural numbers $\mathbb{N} = (\mathbb{N}, +, \cdot, 0, 1)$, and the tropical semirings $\mathbb{R}_{\max} = (\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$ and $\mathbb{R}_{\min} = (\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$.

We say \mathbb{K} is *idempotent* if the addition is idempotent, i.e. $x + x = x$ for all $x \in K$. The semirings \mathbb{B} , \mathbb{R}_{\max} , and \mathbb{R}_{\min} are idempotent. For a general introduction into the theory of semirings and extensive examples, see [11, 19].

Definition 3. A weighted graph automaton (or weighted graph acceptor; wGA) over A, B , and \mathbb{K} is a tuple $\mathcal{A} = (Q, \Delta, \text{wt}, \text{Occ}, r)$ where

- $\mathcal{A}' = (Q, \Delta, \text{Occ}, r)$ is a graph acceptor over the alphabets A and B ,
- $\text{wt} : \Delta \rightarrow K$ is the weight function assigning to every tile of Δ a value of K .

An *accepting run* $\rho : V \rightarrow Q$ of \mathcal{A} on G is defined as an accepting run of \mathcal{A}' on G . Let $\text{sph}_{\mathcal{A}}^r(G_\rho, v)$ be the tile of Δ which is isomorphic to $\text{sph}^r(G_\rho, v)$. We let

$$\text{wt}_{\mathcal{A}, G, \rho}(v) = \text{wt}(\text{sph}_{\mathcal{A}}^r(G_\rho, v)).$$

We define the weight $\text{wt}_{\mathcal{A}, G}(\rho)$ of the run ρ of \mathcal{A} on G as

$$\text{wt}_{\mathcal{A}, G}(\rho) = \prod_{v \in V} \text{wt}_{\mathcal{A}, G, \rho}(v).$$

The behavior $\|\mathcal{A}\| : \text{DG}_t(A, B) \rightarrow K$ of \mathcal{A} is defined, for each $G \in \text{DG}_t(A, B)$, as

$$\|\mathcal{A}\|(G) = \sum_{\rho \text{ accepting run of } \mathcal{A} \text{ on } G} \text{wt}_{\mathcal{A}, G}(\rho).$$

We call any function $S : \text{DG}_t(A, B) \rightarrow K$ a *series*. Then S is *recognizable* if $S = \|\mathcal{A}\|$ for some wGA \mathcal{A} .

By the usual identification of languages with functions assuming values in $\{0, 1\}$, we see that graph acceptors are expressively equivalent to weighted graph automata over the Boolean semiring \mathbb{B} .

Example 4. The following wGA \mathcal{A} counts the number of connected components as exponent of 2. We define $\mathcal{A} = (Q, \Delta, \text{wt}, \text{Occ}, r)$ over arbitrary alphabets A and B , and the semiring $\mathbb{K} = (\mathbb{N}, +, \cdot, 0, 1)$. We set $r = 1$, $\text{Occ} = \text{true}$, $\text{wt} \equiv 1$, and $Q = \{q_1, q_2\}$. The set of tiles is defined as $\Delta = \Delta_1 \cup \Delta_2$, where

$$\Delta_i = \{\tau \mid \text{every vertex of } \tau \text{ is labeled with some } (a, q_i), a \in A\}, i \in \{1, 2\}.$$

Then every connected component of a given graph G is tiled either completely with q_1 or completely with q_2 , thus $\|\mathcal{A}\|(G) = 2^{m(G)}$, where $m(G)$ is the number of connected components of G .

To count occurrences of tiles with a certain pattern, we introduce the following notation. Let τ^* be a finite set of tiles enumerated by $\tau^* = \{\tau_1, \dots, \tau_m\}$. For $N \in \mathbb{N}$, we define the formula

$$\left(\sum_{\tau \in \tau^*} \text{occ}(\tau) \right) \geq N \quad \text{as} \quad \bigvee_{\substack{\sum_{i=1}^m n_i = N \\ n_i \in \{0, \dots, N\}}} \bigwedge_{i=1, \dots, m} \text{occ}(\tau_i) \geq n_i. \quad (1)$$

Using this formula, we can prove the following.

Lemma 5. *Let $S : \text{DG}_t(A, B) \rightarrow K$ be a series recognizable by a wGA \mathcal{A} with tile-size s . Then for all $r \geq s$, S is recognizable by a wGA \mathcal{B} with tile-size r .*

Example 6. Using a weight function which applies the degree of the center of a tile as weight, we can construct a wGA \mathcal{A}_1 satisfying $\|\mathcal{A}_1\|(G) = \prod_{v \in V} \text{degree}(v)$. Using formula (1), we can also construct a wGA \mathcal{A}_2 with $\|\mathcal{A}_2\|(G) = \sum_{v \in V} \text{degree}(v)$. In both cases, we are free to choose a semiring with the desired product or summation, and adjusting wt , we are able to only multiply or sum over vertices of a certain form, e.g., only over vertices labeled with a .

Example 7. The following wGA computes the ‘weighted diameter’ (i.e. the maximal distance between two vertices) of edge-weighted graphs up to a threshold $N \in \mathbb{N}$. Let A be a finite set and $B = \{1, \dots, N\}$ be the edge labels. Note that, here, we sum over the edge labels when computing shortest paths between vertices.

We define $\mathcal{A} = (Q, \Delta, \text{wt}, \text{Occ}, 1)$ over A, B , and \mathbb{R}_{\max} as follows. We set $Q = \{0, \dots, N\} \times \{0, 1\}$. For a vertex v labeled also with $(q, m) \in Q$, we refer to q as the *state* of v , and say v is *marked* if $m = 1$.

Let $\tau = (H, v)$ be a 1-tile with state k at the center v . Then Δ checks that every vertex connected to v by an edge i has a state between $\max\{0, k - i\}$ and $\min\{k + i, N\}$. Furthermore, Δ checks that whenever $k \notin \{0, N\}$, then there has to be at least one vertex in τ which has state $k - i$. The weight function wt assigns to τ the weight k if v is marked, and 0 otherwise. Finally, Occ checks that we have exactly one vertex y with state 0 and exactly one marked vertex z .

Then we can show by induction that every state of a vertex has to be equal to the distance of this vertex to y up to the threshold N . Furthermore, every

run has the weight of the vertex z . Thus, \mathcal{A} computes the maximum distance between two vertices up to the threshold N , which is our desired property. Here, the distance takes the edge-labels into account, whereas setting $B = \{1\}$ yields the unweighted setting and the *geodesic distance* to y up to the threshold N .

For a class \mathcal{C} of graphs, we call the semiring \mathbb{K} \mathcal{C} -regular if for every $k \in K$, there exists a wGA \mathcal{A}_k with $\|\mathcal{A}_k\|(G) = k$ for every $G \in \mathcal{C}$. We call \mathbb{K} regular if \mathbb{K} is $\text{DG}_t(A, B)$ -regular. We give two easy conditions which ensure regularity of \mathbb{K} .

Lemma 8. *If \mathbb{K} is idempotent, then \mathbb{K} is regular. If \mathcal{C} is a class of pointed graphs, then \mathbb{K} is \mathcal{C} -regular.*

We extend the operations $+$ and \cdot of our semiring to series by defining point-wise $(S+T)(G) = S(G)+T(G)$ and $(S\odot T)(G) = S(G)\cdot T(G)$ for each $G \in \text{DG}_t(A, B)$.

Proposition 9. *The class of recognizable series is closed under $+$ and \odot .*

In the following, we show that recognizable series are closed under projection. Let $h : A' \rightarrow A$ be a mapping between two alphabets. Then h defines naturally a relabeling of graphs from $\text{DG}_t(A', B)$ into graphs from $\text{DG}_t(A, B)$, also denoted by h . Let $S : \text{DG}_t(A', B) \rightarrow K$ be a series. We define $h(S) : \text{DG}_t(A, B) \rightarrow K$ by

$$h(S)(G) = \sum_{G' \in \text{DG}_t(A, B), h(G')=G} S(G'). \quad (2)$$

Proposition 10. *Let $S : \text{DG}_t(A', B) \rightarrow K$ be a recognizable series and $h : A' \rightarrow A$. Then $h(S) : \text{DG}_t(A, B) \rightarrow K$ is recognizable.*

4 A Nivat Theorem for Weighted Graph Automata

In this section, we establish a connection between unweighted recognizable languages and recognizable graph series. Note that a corresponding result for weighted automata on words (cf. [12]) makes crucial use of the possible determinization of every unweighted word automaton. Unfortunately, this is not the case for graph languages. To deal with this problem, we require either the underlying semiring to be idempotent or the considered languages to be recognizable by a one-state graph acceptor. For a similar distinction, see [13].

Let $S : \text{DG}_t(A', B) \rightarrow K$ and $L \subseteq \text{DG}_t(A', B)$. We consider $h : A' \rightarrow A$, and the induced mappings $h : \text{DG}_t(A', B) \rightarrow \text{DG}_t(A, B)$, and $h(S) : \text{DG}_t(A, B) \rightarrow K$, as before, see formula (2). We define the *restriction* $S \cap L : \text{DG}_t(A', B) \rightarrow K$ by letting $(S \cap L)(G) = S(G)$ if $G \in L$ and $(S \cap L)(G) = 0$, otherwise.

Let $g : A' \rightarrow K$ be a map. Let $G \in \text{DG}_t(A', B)$ and let $\text{Lab}_G(v) \in A'$ be the label of a vertex v of G . We define the map $\text{prod} \circ g : \text{DG}_t(A', B) \rightarrow K$ by $(\text{prod} \circ g)(G) = \prod_{v \in V} g(\text{Lab}_G(v))$. So, $\text{prod} \circ g : \text{DG}_t(A', B) \rightarrow K$ is a very particular series obtained by assigning, for a graph $G \in \text{DG}_t(A', B)$, to each vertex a weight (depending only on its label) and then multiplying all these weights.

Let $\mathcal{N}_t(A, B, \mathbb{K})$ comprise all series $S : \text{DG}_t(A, B) \rightarrow K$ for which there exist an alphabet A' , a map $g : A' \rightarrow K$, a map $h : A' \rightarrow A$, and a recognizable language $L \subseteq \text{DG}_t(A', B)$ such that $S = h((\text{prod} \circ g) \cap L)$. We denote by $\mathcal{N}_t^{\text{one}}(A, B, \mathbb{K})$ the set of series defined similarly but with a language L which is recognizable by a one-state GA. Trivially, $\mathcal{N}_t^{\text{one}}(A, B, \mathbb{K}) \subseteq \mathcal{N}_t(A, B, \mathbb{K})$.

Using closure properties of series, we get the following Nivat-Theorem for weighted graph automata.

Theorem 11. *Let \mathbb{K} be a commutative semiring and $S : \text{DG}_t(A, B) \rightarrow K$ be a series. Then S is recognizable if and only if $S \in \mathcal{N}_t^{\text{one}}(A, B, \mathbb{K})$. If \mathbb{K} is idempotent, then S is recognizable if and only if $S \in \mathcal{N}_t(A, B, \mathbb{K})$.*

Proof (sketch). First, let S be recognizable by the wGA $\mathcal{A} = (Q, \Delta, \text{wt}, \text{Occ}, r)$ over A, B , and \mathbb{K} . We set $A' = A \times Q \times \text{wt}(\Delta)$. Let h be the projection of A' to A and let g be the projection of A' to $\text{wt}(\Delta)$.

Let $L \subseteq \text{DG}_t(A', B)$ be the language consisting of all graphs G' over A' and B such that assigning to every vertex v' of G' the second component of the label of v' defines an accepting run of \mathcal{A} on $h(G')$ and the added weights are consistent with the weight function wt of \mathcal{A} . We can construct a one-state GA accepting L . It follows that $S = h((\text{prod} \circ g) \cap L) \in \mathcal{N}_t^{\text{one}}(A, B, \mathbb{K})$.

For the converse, let A' be an alphabet, $g : A' \rightarrow K$, $h : A' \rightarrow A$, $L \subseteq \text{DG}_t(A', B)$ be a recognizable language, and $S = h((\text{prod} \circ g) \cap L)$. We can construct a wGA \mathcal{C} which simulates $\text{prod} \circ g$. Using that L is recognizable by a one-state GA or that \mathbb{K} is idempotent, we construct a wGA \mathcal{B} with $\|\mathcal{B}\| = \mathbb{1}_L$. Then Propositions 9 and 10 yield the result. \square

The following lemma and example show that in the general setting there exist series in $\mathcal{N}_t(A, B, \mathbb{K})$ which are not recognizable. For this purpose, we say that a GA \mathcal{A} is *unambiguous* if for every graph G , \mathcal{A} has at most one accepting run on G . We call a graph language L *unambiguously recognizable* if there exists an unambiguous GA accepting L .

We can show that the set of unconnected graphs is a recognizable language which is not unambiguously recognizable. Showing that for every language which is not unambiguously recognizable, the series $\mathbb{1}_L$ is not recognizable over \mathbb{N} , we obtain:

Lemma 12.

1. *The class of unambiguously recognizable languages is a proper subclass of all recognizable languages.*
2. *There exists a recognizable language L such that $\mathbb{1}_L$ is not recognizable over the semiring of the natural numbers \mathbb{N} .*

Example 13. Using ideas of [13], we construct a series in $\mathcal{N}_t(A, B, \mathbb{K})$ which is not recognizable, as follows. We let $\mathbb{K} = (\mathbb{N}, +, \cdot, 0, 1)$, $A' = A$, h be the identity function, and $g \equiv 1$ be the constant function to 1. By Part 2 of Lemma 12, let L be a recognizable language such that $\mathbb{1}_L$ is not recognizable. Then $\mathbb{1}_L = h(\text{prod} \circ g \cap L) \in \mathcal{N}_t(A, B, \mathbb{K})$.

There also exist recognizable but not unambiguously recognizable languages if we consider connected graphs. Over the class of pictures, the existence of such a language was shown in [2].

5 Weighted Logics for Graphs

In the following, we introduce a weighted MSO logic for graphs, following the approach of Droste and Gastin [9] for words. We also incorporate an idea of Bollig and Gastin [4] to consider Boolean formulas.

Definition 14. We define the weighted logic $\text{MSO}(\mathbb{K}, \text{DG}_t(A, B))$, short $\text{MSO}(\mathbb{K})$, as

$$\begin{aligned} \beta &::= P_a(x) \mid E_b(x, y) \mid x = y \mid x \in X \mid \neg\beta \mid \beta \vee \beta \mid \exists x.\beta \mid \exists X.\beta \\ \varphi &::= \beta \mid k \mid \varphi \oplus \varphi \mid \varphi \otimes \varphi \mid \bigoplus_x \varphi \mid \bigoplus_X \varphi \mid \bigotimes_x \varphi \end{aligned}$$

where $k \in K$; x, y are first-order variables; and X is a second order variable.

In [9], the weighted connectors were also denoted by $\vee, \wedge, \exists x, \exists X$, and $\forall x$. We employ this symbolic change to stress the quantitative evaluation of formulas.

Let $G \in \text{DG}_t(A, B)$ and $\varphi \in \text{MSO}(\mathbb{K})$. We follow classical approaches for logics and semantics. Let $\text{free}(\varphi)$ be the set of all free variables in φ , and let \mathcal{V} be a finite set of variables containing $\text{free}(\varphi)$. A (\mathcal{V}, G) -assignment σ is a function assigning to every first-order variable of \mathcal{V} an element of V and to every second order variable a subset of V . We define $\sigma[x \rightarrow v]$ as the $(\mathcal{V} \cup \{x\}, G)$ -assignment mapping x to v and equaling σ everywhere else. The assignment $\sigma[X \rightarrow I]$ is defined analogously.

We represent the graph G together with the assignment σ as a graph (G, σ) over the vertex alphabet $A_{\mathcal{V}} = A \times \{0, 1\}^{\mathcal{V}}$ where 1 denotes every position where x resp. X holds. A graph over $A_{\mathcal{V}}$ is called *valid*, if every first-order variable is assigned to exactly one position.

We define the *semantics* of $\varphi \in \text{MSO}(\mathbb{K})$ as a function $\llbracket \varphi \rrbracket_{\mathcal{V}} : \text{DG}_t(A_{\mathcal{V}}, B) \rightarrow K$ inductively for all valid $(G, \sigma) \in \text{DG}_t(A_{\mathcal{V}}, B)$, as seen in Fig. 1. For not valid (G, σ) , we set $\llbracket \varphi \rrbracket_{\mathcal{V}}(G, \sigma) = 0$. We write $\llbracket \varphi \rrbracket$ for $\llbracket \varphi \rrbracket_{\text{free}(\varphi)}$.

Whether a graph is valid can be checked by an FO-formula, hence the language of all valid graphs over $A_{\mathcal{V}}$ is recognizable. For the Boolean semiring \mathbb{B} , the unweighted MSO is expressively equivalent to $\text{MSO}(\mathbb{B})$.

Lemma 15. Let $\varphi \in \text{MSO}(\mathbb{K})$ and \mathcal{V} a finite set of variables with $\mathcal{V} \supseteq \text{free}(\varphi)$. Then $\llbracket \varphi \rrbracket_{\mathcal{V}}(G, \sigma) = \llbracket \varphi \rrbracket(G, \sigma \upharpoonright \text{free}(\varphi))$ for each valid $(G, \sigma) \in \text{DG}_t(A_{\mathcal{V}}, B)$. Furthermore, if $\llbracket \varphi \rrbracket$ is recognizable, then $\llbracket \varphi \rrbracket_{\mathcal{V}}$ is recognizable.

We note that, in contrast to all previous papers of the literature on weighted logic, in general, the converse of the second statement of Lemma 15 is not true. If we restrict ourselves to pointed graphs or to an idempotent semiring, we can show that $\llbracket \varphi \rrbracket$ is recognizable if and only if $\llbracket \varphi \rrbracket_{\mathcal{V}}$ is recognizable. However, to prove the following statements, the implication above together with Proposition 10 suffices.

Lemma 16. *Let $\llbracket \varphi \rrbracket$ be recognizable. Then $\llbracket \bigoplus_x \varphi \rrbracket$ and $\llbracket \bigoplus_X \varphi \rrbracket$ are recognizable.*

The difficult case is the \bigotimes_x -quantification (previously the universal quantification [9]). Similarly to [9], our unrestricted logic is strictly more powerful than our automata model. Therefore, we introduce the following fragment.

We call a formula $\varphi \in \text{MSO}(\mathbb{K})$ *almost FO-boolean* if φ is built up inductively from unweighted FO-formulas β and constants k using the connectives \oplus and \otimes .

Proposition 17. *Let φ be almost FO-boolean. Then $\llbracket \bigotimes_x \varphi \rrbracket$ is recognizable.*

Proof (sketch). We follow the proof for the universal quantification for words [9] with the crucial difference that we cannot determinize a GA. Let φ be almost FO-boolean. We can show that φ is semantically equivalent to $(k_1 \otimes \varphi_1) \oplus \dots \oplus (k_m \otimes \varphi_m)$, where $m \in \mathbb{N}$, $k_i \in K$, and φ_i are unweighted FO-formulas. Using an extended alphabet $\tilde{A} = A \times \{1, \dots, m\}$, we can encode the finitary structure of $\llbracket \varphi \rrbracket$ into a language \tilde{L} . Since all φ_i are FO-formulas, we can show that \tilde{L} is FO-definable.

Applying Part 1 of Theorem 2, we get a one-state (in particular an unambiguous) GA $\tilde{\mathcal{A}}$, accepting \tilde{L} . We transform $\tilde{\mathcal{A}}$ into a wGA \mathcal{A} which adds the weight defined by the second component of \tilde{A} to the tiles. Then we can show that $\|\mathcal{A}\| = \llbracket \bigotimes_x \varphi \rrbracket$. □

Let $\varphi \in \text{MSO}(\mathbb{K})$. We call φ *restricted* if all unweighted subformulas β are EMSO-formulas and for all subformulas $\bigotimes_x \psi$ of φ , ψ is almost FO-boolean. We call φ *FO-restricted*, if φ is restricted and all unweighted subformulas β are FO-formulas. For the Boolean semiring \mathbb{B} , the unweighted EMSO is expressively equivalent to restricted $\text{MSO}(\mathbb{B})$.

Note that, contrary to [9], we cannot relax our restriction to include unweighted EMSO-formulas after \bigotimes_x because then our restricted weighted logic would still be strictly stronger than EMSO, even for the Boolean semiring.

We can show that every weighted graph automaton can be simulated by an FO-restricted $\text{MSO}(\mathbb{K})$ -sentence. Together with the closure properties of this section and Sect. 3, we obtain our main result.

$$\begin{aligned}
 \llbracket \beta \rrbracket_{\mathcal{V}}(G, \sigma) &= \begin{cases} 1, & \text{if } (G, \sigma) \models \beta \\ 0, & \text{otherwise} \end{cases} & \llbracket k \rrbracket_{\mathcal{V}}(G, \sigma) &= k \quad \text{for all } k \in K \\
 \llbracket \varphi \oplus \psi \rrbracket_{\mathcal{V}}(G, \sigma) &= \llbracket \varphi \rrbracket_{\mathcal{V}}(G, \sigma) + \llbracket \psi \rrbracket_{\mathcal{V}}(G, \sigma) \\
 \llbracket \varphi \otimes \psi \rrbracket_{\mathcal{V}}(G, \sigma) &= \llbracket \varphi \rrbracket_{\mathcal{V}}(G, \sigma) \cdot \llbracket \psi \rrbracket_{\mathcal{V}}(G, \sigma) \\
 \llbracket \bigoplus_x \varphi \rrbracket_{\mathcal{V}}(G, \sigma) &= \sum_{v \in V} \llbracket \varphi \rrbracket_{\mathcal{V} \cup \{x\}}(G, \sigma[x \rightarrow v]) \\
 \llbracket \bigoplus_X \varphi \rrbracket_{\mathcal{V}}(G, \sigma) &= \sum_{I \subseteq V} \llbracket \varphi \rrbracket_{\mathcal{V} \cup \{X\}}(G, \sigma[X \rightarrow I]) \\
 \llbracket \bigotimes_x \varphi \rrbracket_{\mathcal{V}}(G, \sigma) &= \prod_{v \in V} \llbracket \varphi \rrbracket_{\mathcal{V} \cup \{x\}}(G, \sigma[x \rightarrow v])
 \end{aligned}$$

Fig. 1. Semantics

Theorem 18. *Let $\mathbb{K} = (K, +, \cdot, 0, 1)$ be a commutative and regular semiring, and let $S : \text{DG}_t(A, B) \rightarrow K$ be a series. Then the following are equivalent:*

1. S is recognizable.
2. S is definable by an FO-restricted MSO(\mathbb{K})-sentence.

If \mathbb{K} is idempotent, then 1. and 2. are equivalent to

3. S is definable by a restricted MSO(\mathbb{K})-sentence.

Note that to ensure regularity of \mathbb{K} , it suffices to assume \mathbb{K} to be idempotent (for instance as in the case of the tropical semirings) or to consider only pointed graphs (cf. Lemma 8). Words, trees, pictures, and nested words can be seen as pointed structures.

6 Words, Trees, and Other Structures

In this section, using ideas from Thomas [34], we show that existing quantitative automata models over words [9], trees [14], pictures [17], and nested words [24] can be seen as special incidences of weighted graph automata with the same expressive power. Hence, we get previous equivalence results connecting weighted logic and weighted automata over these structures as a consequence (in a slightly modified version).

As already stated by [34], two significant differences to the previous models are the occurrence constraint and the possibly bigger tile-size. As first step, following [34], we give a sufficient condition to drop the occurrence constraint. We say that a weighted graph automaton $\mathcal{A} = (Q, \Delta, \text{wt}, \text{Occ}, r)$ is *without occurrence constraint*, if $\text{Occ} = \text{true}$.

We call a class of graphs $\mathcal{C} \subseteq \text{DG}_t(A, B)$ *partial sortable* if there exists an element $b_0 \in B$ such that for every graph $G = (V, (P_a)_{a \in A}, (E_b)_{b \in B})$ of \mathcal{C} , the subgraph $G' = (V, (P_a)_{a \in A}, E_{b_0})$ is acyclic, connected, and every vertex of G' has at most one outgoing edge.

Note that words, trees, pictures, and nested words can be seen as graph classes of this type. The following result is a weighted (and slightly more general) version of Proposition 5.3 in [34].

Lemma 19. *Let $\mathcal{C} \subseteq \text{DG}_t(A, B)$ be partial sortable and $S : \mathcal{C} \rightarrow K$ a recognizable series. Then there exists a wGA \mathcal{B} without occurrence constraint with $\|\mathcal{B}\| = S$.*

Definition 20. *A weighted finite automata (wFA) $\mathcal{A} = (Q, I, F, \delta, \mu)$ consists of a finite set of states Q , a set of initial states $I \subseteq Q$, a set of final states $F \subseteq Q$, a set of transitions $\delta \subseteq Q \times A \times Q$, and a weight function $\mu : \delta \rightarrow K$. We define an accepting run, the language of \mathcal{A} , and recognizable word series as usual (cf. [9, 11, 30]).*

Now we consider words (trees, pictures, nested words, respectively) as relational structures, and hence also as graphs. Using Lemma 19, we can prove the following.

Proposition 21. *Let $S : A^* \rightarrow K$ be a word series. Then S is recognizable by a weighted graph automaton iff S is recognizable by a weighted finite automaton.*

A similar result can be proved for trees, pictures, and nested words, and their respective automata models defining recognizable series.

Note that this is possible because, essentially, we can reduce the tile-size of a weighted graph automaton over these specific graphs from r to 1. In general, this is not possible, and it is already stated by Thomas [34] as an open question to precisely describe the class of graphs where the use of 1-tiles suffices.

We obtain the following consequence of Theorem 18 and Proposition 21.

Corollary 22 ([9]). *For a word series $S : A^* \rightarrow K$ the following are equivalent:*

1. *S is recognizable by a weighted finite automaton.*
2. *S is definable by an FO-restricted MSO(\mathbb{K})-sentence.*

If we replace “word” by “tree”, “picture”, or “nested word”, we get a similar result for all four automata models and their respective logics.

Note that our implication (2) \Rightarrow (1) is a slightly weaker version of the results in [9] since in our logic, we can apply \bigotimes_x -quantification (resp. universal quantification) only to almost FO-boolean formulas, whereas in [9, 10], we can use almost MSO-boolean formulas.

As shown before, this difference originates from the fact that in the word case EMSO = MSO, which is not true for pictures or graphs. In this sense, our result captures the ‘biggest logic possible’, if we want to include pictures.

7 Conclusion

We introduced a weighted generalization of Thomas’ graph acceptors [33, 34] and a suitable weighted logic in the sense of Droste and Gastin [9]. For commutative semirings, we proved a Nivat-like and a Büchi-like characterization of the expressive power of weighted graph automata. We showed slightly stronger results in the case of an idempotent semiring.

We showed that weighted word, tree, picture, or nested word automata are special instances of these general weighted graph automata, which gives us results of [9, 14, 17, 24] as corollaries (under the appropriate restrictions to the underlying logic). Although not considered explicitly, we conjecture that similar equivalence results also hold for other finite structures like traces [25] or texts [23] and their respective automata models. We gave several examples that our weighted graph automata can recognize particular quantitative properties of graphs which were not covered by previous automata models.

Most statements in this paper can also be proven for general semirings. In this case, however, we would have to enforce an ordering of our graphs to get a well-defined product of weights. Furthermore, we would need some restrictions on the conjunction of our logic (similar to, e.g., [10]). Here, we restricted ourselves to commutative semirings in order to avoid technical conditions.

All the constructions given in this paper are effective. Subsequent research could investigate applications of weighted graph algorithms for developing decision procedures for weighted graph automata.

References

1. Alur, R., Madhusudan, P.: Adding nesting structure to words. *J. ACM* **56**(3), 16:1–16:43 (2009)
2. Anselmo, M., Giammarresi, D., Madonia, M., Restivo, A.: Unambiguous recognizable two-dimensional languages. *ITA* **40**(2), 277–293 (2006)
3. Berstel, J., Reutenauer, C.: *Rational Series and Their Languages*. EATCS Monographs in Theoretical Computer Science, vol. 12. Springer, Heidelberg (1988)
4. Bollig, B., Gastin, P.: Weighted versus probabilistic logics. In: Diekert, V., Nowotka, D. (eds.) *DLT 2009*. LNCS, vol. 5583, pp. 18–38. Springer, Heidelberg (2009)
5. Bollig, B., Gastin, P., Monmege, B., Zeitoun, M.: Pebble weighted automata and weighted logics. *ACM Trans. Comput. Log.* **15**(2), 15 (2014)
6. Büchi, J.R.: Weak second-order arithmetic and finite automata. *Z. Math. Logik und Grundlagen Math.* **6**, 66–92 (1960)
7. Doner, J.: Tree acceptors and some of their applications. *J. Comput. Syst. Sci.* **4**(5), 406–451 (1970)
8. Droste, M., Dück, S.: Weighted automata and logics for infinite nested words. In: Dediu, A.-H., Martín-Vide, C., Sierra-Rodríguez, J.-L., Truthe, B. (eds.) *LATA 2014*. LNCS, vol. 8370, pp. 323–334. Springer, Heidelberg (2014)
9. Droste, M., Gastin, P.: Weighted automata and weighted logics. *Theor. Comput. Sci.* **380**(1–2), 69–86 (2007)
10. Droste, M., Gastin, P.: Weighted automata and weighted logics. In: Droste et al. [11], chapter 5, pp. 175–211
11. Droste, M., Kuich, W., Vogler, H. (eds.): *Handbook of Weighted Automata*. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg (2009)
12. Droste, M., Kuske, D.: Weighted automata. In: Pin, J.E. (ed.) *Handbook: “Automata: from Mathematics to Applications”*. Europ. Mathematical Soc. (to appear)
13. Droste, M., Perevoshchikov, V.: A Nivat theorem for weighted timed automata and weighted relative distance logic. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) *ICALP 2014, Part II*. LNCS, vol. 8573, pp. 171–182. Springer, Heidelberg (2014)
14. Droste, M., Vogler, H.: Weighted tree automata and weighted logics. *Theor. Comput. Sci.* **366**(3), 228–247 (2006)
15. Eilenberg, S.: *Automata, Languages, and Machines*, Pure and Applied Mathematics, vol. 59-A. Academic Press, New York (1974)
16. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. *Trans. Am. Math. Soc.* **98**(1), 21–52 (1961)
17. Fichtner, I.: Weighted picture automata and weighted logics. *Theory Comput. Syst.* **48**(1), 48–78 (2011)
18. Giammarresi, D., Restivo, A., Seibert, S., Thomas, W.: Monadic second-order logic over rectangular pictures and recognizability by tiling systems. *Inf. Comput.* **125**(1), 32–45 (1996)

19. Golan, J.S.: Semirings and their Applications. Kluwer Academic Publishers, Dordrecht (1999)
20. Hanf, W.: Model-theoretic methods in the study of elementary logic. In: Addison, J., Henkin, L., Tarski, A. (eds.) *The Theory of Models*, pp. 132–145. Amsterdam, North-Holland (1965)
21. Hoogeboom, H.J., ten Pas, P.: Monadic second-order definable text languages. *Theory Comput. Syst.* **30**(4), 335–354 (1997)
22. Kuich, W., Salomaa, A.: Semirings, Automata, Languages. EATCS Monographs in Theoretical Computer Science, vol. 6. Springer, Heidelberg (1986)
23. Mathissen, C.: Definable transductions and weighted logics for texts. *Theor. Comput. Sci.* **411**(3), 631–659 (2010)
24. Mathissen, C.: Weighted logics for nested words and algebraic formal power series. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008, Part II*. LNCS, vol. 5126, pp. 221–232. Springer, Heidelberg (2008)
25. Meinecke, I.: Weighted logics for traces. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) *CSR 2006*. LNCS, vol. 3967, pp. 235–246. Springer, Heidelberg (2006)
26. Monmege, B.: *Specification and Verification of Quantitative Properties: Expressions, Logics, and Automata*. Thèse de doctorat, ENS Cachan, France (2013)
27. Nivat, M.: Transductions des langages de Chomsky. *Ann. de l'Inst. Fourier* **18**, 339–455 (1968)
28. Rabin, M.O.: Decidability of second order theories and automata on infinite trees. *Trans. Am. Math. Soc.* **141**, 1–35 (1969)
29. Salomaa, A., Soittola, M.: *Automata-Theoretic Aspects of Formal Power Series*. Texts and Monographs in Computer Science. Springer, New York (1978)
30. Schützenberger, M.P.: On the definition of a family of automata. *Inf. Control* **4**(2–3), 245–270 (1961)
31. Thatcher, J.W., Wright, J.B.: Generalized finite automata theory with an application to a decision problem of second-order logic. *Math. Syst. Theory* **2**(1), 57–81 (1968)
32. Thomas, W.: On logical definability of trace languages. In: Diekert, V. (ed.) *Proceedings of workshop ASMICS 1989*, pp. 172–182. Technical University of Munich (1990)
33. Thomas, W.: On logics, tilings, and automata. In: Leach Albert, J., Monien, B., Rodríguez-Artalejo, M. (eds.) *ICALP 1991*. LNCS, vol. 510, pp. 441–454. Springer, Heidelberg (1991)
34. Thomas, W.: Elements of an automata theory over partial orders. In: *Proceedings of DIMACS Workshop POMIV 1996*, pp. 25–40. AMS Press Inc, New York, USA (1996)
35. Trakhtenbrot, B.A.: Finite automata and logic of monadic predicates (in Russian). *Doklady Akademii Nauk SSR* **140**, 326–329 (1961)

Longest Gapped Repeats and Palindromes

Marius Dumitran¹ and Florin Manea²(✉)

¹ Faculty of Mathematics and Computer Science,
University of Bucharest, Academiei 14, 010014 Bucharest, Romania
`marius.dumitran@fmi.unibuc.ro`

² Department of Computer Science, Christian-Albrechts University of Kiel,
Christian-Albrechts-Platz 4, 24118 Kiel, Germany
`flm@informatik.uni-kiel.de`

Abstract. A gapped repeat (respectively, palindrome) occurring in a word w is a factor uvu (respectively, u^Rvu) of w . We show how to compute efficiently, for every position i of the word w , the longest prefix u of $w[i..n]$ such that uv (respectively, u^Rv) is a suffix of $w[1..i-1]$ (defining thus a gapped repeat uvu – respectively, palindrome u^Rvu), and the length of v is subject to various types of restrictions.

1 Introduction

Gapped repeats and palindromes have been investigated for a long time (see, e.g., [1–7] and the references therein), with motivation coming especially from the analysis of DNA and RNA structures, where tandem repeats or hairpin structures play important roles in revealing structural and functional information of the analysed genetic sequence (see [1, 2, 4] and the references therein). More precisely, a gapped repeat (respectively, palindrome) occurring in a word w is a factor uvu (respectively, u^Rvu) of w . The middle part v of such structures is called gap, while the two factors u (or the factors u^R and u) are called left and right arms. Generally, the previous works were interested in finding all the gapped repeats and palindromes, under certain restrictions on the length of the gap or on the relation between the arm of the repeat or palindrome and the gap.

In this paper, we propose an alternative point of view in the study of gapped repeats and palindromes. The longest previous factor table (LPF) was introduced and considered in the context of efficiently computing Lempel-Ziv-like factorisations of words (see [6, 8]). Such a table provides for each position i of the word the longest factor occurring both at position i and once again on a position $j < i$. Several variants of this table were also considered in [6]: the longest previous reverse factor (*LPrF*), where we look for the longest factor occurring at position i and whose mirror image occurs in the prefix of w of length $i-1$, or the longest previous non-overlapping factor, where we look for the longest factor occurring both at position i and somewhere inside the prefix of length $i-1$ of w . Such tables may be seen as providing a comprehensive image of the long repeats and symmetries occurring in the analysed word. In our work we approach the construction of longest previous gapped repeat or palindrome tables: for each

position i of the word we want to compute the longest factor occurring both at position i and once again on a position $j < i$ (or, respectively, whose mirror image occurs in the prefix of length $i - 1$ of w) such that there is a gap (subject to various restrictions) between i and the previous occurrence of the respective factor (mirrored factor). Similar to the original setting, this should give us a good image of the long gapped repeats and symmetries of a word.

A simple way to restrict the gap is to lower bound it by a constant; i.e., we look for factors uvu (or u^Rvu) with $|v| > g$ for some $g \geq 0$. The techniques of [6] can be easily adapted to compute for each position i the longest prefix u of $w[i..n]$ such that there exists a suffix uv (respectively, u^Rv) of $w[1..i - 1]$, forming thus a factor uvu (respectively, u^Rvu) with $|v| > g$. Here we consider three other different types of restricted gaps.

We first consider the case when the length of the gap is between a lower bound g and an upper bound G , where g and G are given as input (so, may depend on the input word). This extends naturally the case of lower bounded gaps.

Problem 1. Given w of length n and two integers g and G , such that $0 \leq g < G \leq n$, construct the arrays $LPrF_{g,G}[\cdot]$ and $LPF_{g,G}[\cdot]$ defined for $1 \leq i \leq n$:

- a. $LPrF_{g,G}[i] = \max\{|u| \mid \text{there exists } v \text{ such that } u^Rv \text{ is a suffix of } w[1..i - 1] \text{ and } u \text{ is prefix of } w[i..n], \text{ with } g \leq |v| < G\}$.
- b. $LPF_{g,G}[i] = \max\{|u| \mid \text{there exists } v \text{ such that } uv \text{ is a suffix of } w[1..i - 1] \text{ and } u \text{ is prefix of } w[i..n], \text{ with } g \leq |v| < G\}$.

We are able to solve Problem 1(a) in linear time $\mathcal{O}(n)$. Problem 1(b) is solved here in $\mathcal{O}(n \log n)$ time. Intuitively, when trying to compute the longest prefix u of $w[i..n]$ such that u^Rv is a suffix of $w[1..i - 1]$ with $g < |v| \leq G$, we just have to compute the longest common prefix between $w[i..n]$ and the words $w[1..j]^R$ with $g < i - j \leq G$. The increased difficulty in solving the problem for repeats (reflected in the increased complexity of our algorithm) seems to come from the fact that when trying to compute the longest prefix u of $w[i..n]$ such that uv is a suffix of $w[1..i - 1]$ with $g < |v| \leq G$, it is hard to see where the uv factor may start, so we have to somehow try more variants for the length of u . In [2], the authors give an algorithm that finds all maximal repeats (i.e., repeats whose arms cannot be extended) with gap between a lower and an upper bound, running in $\mathcal{O}(n \log n + z)$ time, where z is the number of such repeats. It is worth noting that there are words (e.g., $(a^2b)^{n/3}$, from [2]) that may have $\Theta(nG)$ maximal repeats uvu with $|v| < G$, so for $G > \log n$ and $g = 0$, for instance, our algorithm is faster than an approach that would first use the algorithms of [2] to get all maximal repeats, and then process them somehow to solve Problem 1(b).

In the second case, the gaps are only lower bounded; however, the bound on the gap allowed at a position is defined by a function depending on that position.

Problem 2. Given w of length n and the values $g(1), \dots, g(n)$ of $g : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$, construct the arrays $LPrF_g[\cdot]$ and $LPF_g[\cdot]$ defined for $1 \leq i \leq n$:

- a. $LPrF_g[i] = \max\{|u| \mid \text{there exists } v \text{ such that } u^Rv \text{ is a suffix of } w[1..i - 1] \text{ and } u \text{ is prefix of } w[i..n], \text{ with } g(i) \leq |v|\}$.

- b. $LPF_g[i] = \max\{|u| \mid \text{there exists } v \text{ such that } uv \text{ is a suffix of } w[1..i-1] \text{ and } u \text{ is prefix of } w[i..n], \text{ with } g(i) \leq |v|\}.$

The setting of this problem can be seen as follows. An expert preprocesses the input word (in a way specific to the framework in which one needs the problem solved), and detects the length of the gap occurring at each position (so, computes $g(i)$ for all i). These values and the word are then given to us, to compute the arrays defined in our problems. We solve both problems in linear time.

Finally, following [4, 5], we analyse gapped repeats uvu and palindromes u^Rvu where the length of the gap v is upper bounded by the length of the arm u ; these structures are called long armed repeats and palindromes, respectively.

Problem 3. Given w of length n , construct the arrays $LPal[\cdot]$ and $LRep[\cdot]$, defined for $1 \leq i \leq n$:

- a. $LPal[i] = \max\{|u| \mid \text{there exists } v \text{ such that } u^Rv \text{ is a suffix of } w[1..i-1], u \text{ is a prefix of } w[i..n], \text{ and } |v| \leq |u|\}.$
- b. $LRep[i] = \max\{|u| \mid \text{there exists } v \text{ such that } uv \text{ is a suffix of } w[1..i-1], u \text{ is a prefix of } w[i..n], \text{ and } |v| \leq |u|\}.$

In [4] one proposes an algorithm finding the set S of all factors of a word of length n which are maximal long armed palindromes (i.e., the arms cannot be extended to the right or to the left) in $\mathcal{O}(n + |S|)$ time; no upper bound on the possible size of the set S was given in [4], but it is widely believed to be $\mathcal{O}(n)$. Using the algorithm of [4] as an initial step, we solve Problem 3(a) in $\mathcal{O}(n + |S|)$ time. In [5] the set of maximal long armed repeats with non-empty gap is shown to be of linear size and is computed in $\mathcal{O}(n)$ time. We use this algorithm and a linear time algorithm finding the longest square centred at each position of a word to solve Problem 3(b) in linear time.

Our algorithms are generally based on efficient data-structures. On one hand, we use efficient word-processing data structures like suffix arrays or longest common prefix structures. On the other hand, we heavily use specific data-structures for maintaining efficiently collections of disjoint sets, under union and find operations. Alongside these data-structures, we make use of a series of remarks of combinatorial nature, providing insight in the repetitive structure of the words.

2 Preliminaries

Let V be a finite alphabet; V^* denotes the set of all finite words over V . The *length* of a word $w \in V^*$ is denoted by $|w|$. The *empty word* is denoted by λ . A word $u \in V^*$ is a *factor* of $v \in V^*$ if $v = xuy$, for some $x, y \in V^*$; we say that u is a *prefix* of v , if $x = \lambda$, and a *suffix* of v , if $y = \lambda$. We denote by $w[i]$ the symbol occurring at position i in w , and by $w[i..j]$ the factor of w starting at position i and ending at position j , consisting of the catenation of the symbols $w[i], \dots, w[j]$, where $1 \leq i \leq j \leq n$; we define $w[i..j] = \lambda$ if $i > j$. The powers of a word w are defined recursively by $w^0 = \lambda$ and $w^n = ww^{n-1}$ for $n \geq 1$.

If w cannot be expressed as a nontrivial power (i.e., w is not a repetition) of another word, then w is *primitive*. A *period* of a word w over V is a positive integer p such that $w[i] = w[j]$ for all i and j with $i \equiv j \pmod{p}$. Let $\mathbf{per}(w)$ be the smallest period of w . A word w with $\mathbf{per}(w) \leq \frac{|w|}{2}$ is called run; a run $w[i..j]$ (so, $p = \mathbf{per}(w[i..j]) < \frac{j-i+1}{2}$) is maximal iff it cannot be extended to the left or right to get a word with the same period p , i.e., $i = 1$ or $w[i-1] \neq w[i+p-1]$, and, $j = n$ or $w[j+1] \neq w[j-p+1]$. In [9] it is shown that the number of maximal runs of a word is linear and their list (with a run $w[i..j]$ represented as the triple $(i, j, \mathbf{per}(w[i..j]))$) can be computed in linear time. The exponent of a maximal run $w[i..j]$ occurring in w is defined as $\frac{j-i+1}{\mathbf{per}(w[i..j])}$; the sum of the exponents of all runs in a word of length n is $\mathcal{O}(n)$ (see [9]).

For a word u , $|u| = n$, over $V \subseteq \{1, \dots, n\}$ we build in $\mathcal{O}(n)$ time the suffix array as well as data structures allowing us to retrieve in constant time the length of the longest common prefix of any two suffixes $u[i..n]$ and $u[j..n]$ of u , denoted $LCP_u(i, j)$ (the subscript u is omitted when there is no danger of confusion). Such structures are called *LCP data structures* in the following. For details, see, e.g., [1, 10], and the references therein.

In the solutions of the problems dealing with gapped palindromes inside a word w (Problems 1(a), 2(a), and 3(a)) we construct the suffix array and *LCP* data structures for the word $u = w0w^R$, where 0 is a symbol lexicographically smaller than all the symbols of V ; this takes $\mathcal{O}(|w|)$ time. To check whether $w[i..j]$ occurs at position ℓ in w (respectively, $w[i..j]^R$ occurs at position ℓ in w) we check whether $\ell + (j - i + 1) \leq n$ and $LCP_u(i, \ell) \geq j - i + 1$ (respectively, $LCP_u(\ell, 2|w| - j + w) \geq j - i + 1$). To keep the notation free of subscripts, when we measure the longest common prefix of a word $w[1..j]^R$ and word $w[i..n]$ we write $LCP(w[1..j]^R, w[1..i])$, and this is in fact an *LCP*-query on $u = w0w^R$; when we measure the longest common prefix of a word $w[j..n]$ and word $w[i..n]$ we write $LCP(j, i)$, and this is in fact an *LCP*-query on w .

The suffix array of $w0w^R$ allows us to construct in linear time a list \mathcal{L} of the suffixes $w[i..n]$ of w and of the mirror images $w[1..i]^R$ of the prefixes of w (which correspond to the suffixes of length less than $|w|$ of $w0w^R$), ordered lexicographically. Generally, we denote by $Rank[i]$ the position of $w[i..n]$ in the ordered list of these factors, and by $Rank_R[i]$ the position of $w[1..i]^R$ in \mathcal{L} .

Our solutions rely on an efficient solution for the *disjoint set union-find* problem. This problem asks to maintain a family consisting initially of d disjoint singleton sets from the universe $U = [1, n + 1]$ (shorter for $\{1, \dots, n\}$) so that given any element we can locate its current set and return the minimal (and/or the maximal) element of this set (operation called *find-query*) and we can merge two disjoint sets into one (operation called *union*). In our framework, we know from the beginning the pairs of elements whose corresponding sets can be joined. Under this assumption, a data-structure fulfilling the above requirements can be constructed in $\mathcal{O}(d)$ time such that performing a sequence of m find and union operations takes $\mathcal{O}(m)$ time in our model of computation (see [11]). As a particular case, this data structure solves with $\mathcal{O}(n)$ preprocessing time and $\mathcal{O}(1)$ amortised time per operation the *interval union-find* problem, which asks to

maintain a partition of the universe $U = [1, n + 1)$ into a number of d disjoint intervals, so that given an element of U we can locate its current interval, and we can merge two adjacent intervals of the partition.

In the solutions of both points of Problem 1, we use the following variant of the interval union-find problem.

Remark 1. Let $U = [1, n + 1)$. We are given integers $d, k > 0$ and for each $i \leq n/k$ a partition \mathcal{P}_i of U in d intervals, a sequence of d find-queries and a sequence of d union-operations to be performed alternatively on \mathcal{P}_i (that is, we perform one find query then the one union operation, and so on, in the order specified in the sequences). We can obtain the answers to all find-queries in $\mathcal{O}(n + \frac{nd}{k})$ time.

We further give a lemma related to the union-find data structure.

Lemma 1. *Let $U = [1, n + 1)$. We are given k intervals I_1, \dots, I_k included in U . Also, for each $j \leq k$ we are given a positive integer $g_j \leq n$, the weight of the interval I_j . We can compute in $\mathcal{O}(n + k)$ time the values $H[i] = \max\{g_j \mid i \in I_j\}$ (or, alternatively, $h[i] = \min\{g_j \mid i \in I_j\}$) for all $i \leq n$.*

We conclude this section with a lemma that will be used in the solution of Problem 3. It shows that how can compute in linear time the length of the longest square centred at each position of a given word.

Lemma 2. *Given a word w of length n we can compute in $\mathcal{O}(n)$ time the values $SC[i] = \max\{|u| \mid u \text{ is both a suffix of } w[1..i - 1] \text{ and a prefix of } w[i..n]\}$.*

Proof. Note that each square u^2 occurring in a word w is part of a maximal run $w[i'..j'] = p^\alpha p'$, where p is primitive and p' is a prefix of p , and $u = q^\ell$, where q is a cyclic shift of p (i.e., q is a factor of p^2 of length $|p|$) and $\ell \leq \frac{\alpha}{2}$.

So, if we consider a maximal run $r = p^\alpha p'$ and some $\ell \leq \frac{\alpha}{2}$, we can easily detect the possible centre positions of the squares having the form $(q^\ell)^2$ contained in this run, with q a cyclic shift of p . These positions occur consecutively in the word w : the first is the $(|p|^\ell + 1)^{th}$ position of the run, and the last is the one where the suffix of length $|p|^\ell$ of the run starts. So they form an interval $I_{r,\ell}$ and we associate to this interval the weight $g_{r,\ell} = |p|^\ell$ (i.e., the length of an arm of the square). In this way, we define $\mathcal{O}(n)$ intervals (as their number is upper bounded by the sum of the exponents of the maximal runs of w), all contained in $[1, n + 1)$, and each interval having a weight between 1 and n . By Lemma 1, we can process these intervals so that we can determine for each $i \in [1, n + 1)$ the interval of maximum weight containing i , or, in other words, the maximum length $SC[i]$ of a square centred on i . This procedure runs in $\mathcal{O}(n)$ time. \square

Remark 2. It is worth noting that using the same strategy as in the proof of Lemma 2, one can detect the minimum length of a square centred at each position of a word in $\mathcal{O}(n)$ time. This leads to an alternative solution to the problem of computing the local periods of a word, solved in [12]. Compared to the solution from [12], ours uses a relatively involved data structures machinery (disjoint sets union-find structures), but is much shorter and seems conceptually simpler as it does not require a very long and detailed combinatorial analysis of the properties

of the input word. The same strategy allows solving the problem of computing in linear time, for integer alphabets, the length of the shortest (or longest) square ending (or starting) at each position of a given word; this improves the results from [13,14], where such a result was only shown for constant size alphabets.

3 Lower and Upper Bounded Gap

In this section we approach Problems 1(a) and 1(b). We give the full solution of Problem 1(a), and then point out how this can be extended to solve Problem 1(b).

Theorem 1. *Problem 1(a) can be solved in linear time.*

Proof. Let $\delta = G - g$; for simplicity, assume that n is divisible by δ . Further, for $1 \leq i \leq n$, let u be the longest factor which is a prefix of $w[i..n]$ such that u^R is a suffix of some $w[1..k]$ with $g < i - k \leq G$; then, $B[i]$ is the rightmost position j such that $g < i - j \leq G$ and u^R is a suffix of $w[1..j]$. Knowing $B[i]$ means knowing $LPrF_{g,G}[i]$: we just have to return $LPrF(w[i..n], w[1..j]^R)$.

We split the set $\{1, \dots, n\}$ into $\frac{n}{\delta}$ ranges of consecutive numbers: $I_0 = \{1, 2, \dots, \delta\}$, $I_1 = \{\delta + 1, \delta + 2, \dots, 2\delta\}$, and so on. Note that for all i in some range $I_k = \{k\delta + 1, k\delta + 2, \dots, (k + 1)\delta\}$ from those defined above, there are at most three consecutive ranges where some j such that $g < i - j \leq G$ may be found: the range containing $k\delta - G + 1$, the range containing $(k + 1)\delta - g - 1$, and the one in between these two (i.e., the range containing $k\delta + 1 - g$). Moreover, for some fixed i , we know that we have to look for $B[i]$ in the interval $\{i - G, \dots, i - g - 1\}$, of length δ ; when we search $B[i + 1]$ we look at the interval $\{i + 1 - G, \dots, i - g\}$. So, basically, when trying to find $B[i]$ for all $i \in I_k$, we move a window of length δ over the three ranges named above, and try to find for every content of the window (so for every i) its one element that fits the description of $B[i]$. The difference between the content of the window in two consecutive steps does not seem major: we just removed an element and introduced a new one. Also, note that at each moment the window intersects exactly two of the aforementioned three ranges. We try to use these remarks, and maintain the contents of the window such that the update can be done efficiently, and the values of $B[i]$ (and, implicitly, $LPrF_{g,G}[i]$) can be, for each $i \in I$, retrieved very fast. Intuitively, grouping the i 's on ranges of consecutive numbers allows us to find the possible places of the corresponding $B[i]$'s for all $i \in I_k$ in $\mathcal{O}(\delta)$ time.

We now go into more details. As we described in the preliminaries, let \mathcal{L} be the lexicographically ordered list of the suffixes of $w[i..n]$ of w and of the mirror images $w[1..i]^R$ of the prefixes of w (which correspond to the suffixes of w^R). For the list \mathcal{L} we compute the arrays $Rank[\cdot]$ and $Rank_R[\cdot]$. We use the suffix array for $w0w^R$ to produce for each of the ranges I_k computed above the set of suffixes of w that start in the respective range (sorted lexicographically, in the order they appear in the suffix array) and the set of prefixes of w ending in I , ordered lexicographically with respect to their mirror image.

We consider now one of the ranges of indexes $I_k = \{k\delta + 1, k\delta + 2, \dots, (k + 1)\delta\}$ from the above, and show how we can compute the values $B[i]$, for all $i \in I_k$. For

some $i \in I_k$ we look for the maximum ℓ such that there exists a position j with $w[j - \ell + 1..j]^R = w[i..i + \ell - 1]$, and $i - G \leq j \leq i - g$. As already explained, for the i 's of I_k there are three consecutive ranges where the j 's corresponding to the i 's of I_k may be found. Let us denote them J_1, J_2, J_3 .

Now, for an $i \in I_k$ we have that $B[i]$ (for which $g < i - B[i] \leq G$) belongs to $J'_i \cup J''_i$, where J'_i is an interval starting with $i - G$ and extending until the end of the range that contains $i - G$ (which is one of the ranges J_1, J_2, J_3) and J''_i is an interval ending with $i - g - 1$, which starts at the beginning of the range that contains $i - g - 1$ (which is the range that comes next after the one containing $i - G$). Referencing back to the intuitive explanation we gave at the beginning of this proof, $J'_i \cup J''_i$ is the window we use to locate the value of $B[i]$ for an $i \in I_k$.

To compute $B[i]$ for some i , take $f_i \in J'_i$ such that $LCP(w[1..f_i]^R, w[i..n]) \geq LCP(w[1..j']^R, w[i..n])$ for all $j' \in J'_i$. Similarly, take $s_i \in J''_i$ such that for all $j' \in J''_i$ we have $LCP(w[1..s_i]^R, w[i..n]) \geq LCP(w[1..j']^R, w[i..n])$. Once s_i and f_i computed, we just have to set $B[i] = s_i$ if $LCP(w[1..s_i]^R, w[i..n]) \geq LCP(w[1..f_i]^R, w[i..n])$; we set $B[i] = f_i$, otherwise. So, in order to compute, for some i , the value $B[i]$, that determines $LPrF_{g,G}[i]$, we first compute f_i and s_i .

We compute for all the indices $i \in I_k$, considered in increasing order, the values f_i . We consider for each $i \in I_k$ the interval J'_i and note that $J'_i \setminus J'_{i+1} = \{i - G\}$, and, if J'_i is not a singleton (i.e., $J'_i \neq \{i - G\}$) then $J'_{i+1} \subset J'_i$. If J'_i is a singleton, than J'_{i+1} is, in fact, one of the precomputed range I_p , namely the one which starts on position $i + 1 - G$ (so, $p = \frac{i-G}{\delta}$).

These relations suggest the following approach. We start with $i = k\delta + 1$ and consider the set of words $w[1..j]^R$ with $j \in J'_i$; this set can be easily obtained in $\mathcal{O}(\delta)$ time by finding first the range J_1 in which $i - G$ is contained (which takes $\mathcal{O}(1)$ time, as $J_1 = I_p$ for $p = \lfloor \frac{i-G}{\delta} \rfloor$), and then selecting from J_1 of the set of prefixes $w[1..d]$ of w , ending in J_1 with $d \geq i - G$ (ordered lexicographically with respect to their mirror image). The ranks corresponding to these prefixes in the ordered list \mathcal{L} (i.e., the set of numbers $Rank_R[d]$) define a partition of the universe $U = [0, 2n + 1]$ in at most $\delta + 2$ disjoint intervals. So, we can maintain an interval union-find data structures like in Remark 1, where the ranks are seen as limits of the intervals in this structure. We assume that the intervals in our data structure are of the form $[a, b)$, with a and b equal to some $Rank_R[d_a]$ and $Rank_R[d_b]$, respectively. The first interval in the structure is of the form $[0, a)$, while the last is of the form $[b, 2n + 1)$. We now find the interval to which $Rank[i]$ belongs; say that this is $[a, b)$. This means that the words $w[1..d_a]^R$ and $w[1..d_b]^R$ are the two words of $\{w[1..d]^R \mid d \in J'_i\}$ which are closest to $w[i..n]$ lexicographically ($w[1..d_a]^R$ is lexicographically smaller, $w[1..d_b]^R$ is greater). Clearly, $f_i = d_a$ if $LCP(w[1..d_a]^R, w[i..n]) \geq LCP(w[1..d_b]^R, w[i..n])$ and $f_i = d_b$, otherwise (in case of a tie, we take f_i to be the greater of d_a and d_b). So, to compute f_i we query once the union-find data structure to find a and b , and the corresponding d_a and d_b , and then run two more LCP queries.

When moving on to compute f_{i+1} , we just have to update our structure and then run the same procedure. Now, $i - G$ is no longer a valid candidate for f_{i+1} , and it is removed from J'_i . So we just delete it from the interval union-find

data structure, and merge the interval ending right before $Rank_R[i - G]$ and the one starting with $Rank_R[i - G]$. This means one union operation in our interval union-find structure. Then we proceed to compute f_{i+1} as in the case of f_i .

The process continues until J'_i is a singleton, so f_i equals its single element.

Now, $i - G$ is the last element of one of the ranges J_1, J_2 , or J_3 ; assume this range is I_p . So far, we performed alternatively at most δ find queries and δ union operations on the union-find structure. Now, instead of updating this structure, we consider a new interval partition of $[0, 2n + 1)$ induced by the ranks of the δ prefixes ending in I_{p+1} . When computing the values f_i for $i \in I_k$ we need to consider a new partition of U at most once: at the border between J_1 and J_2 .

It is not hard to see from the comments made in the above presentation that our algorithm computes $f_i \in I_k$ correctly. In summary, in order to compute f_i , we considered the elements $i \in I_k$ from left to right, keeping track of the left part J'_i of the window $J'_i \cup J''_i$ while it moved from left to right through the ranges J_1, J_2 and J_3 . Now, to compute the values s_i for $i \in I_k$, we proceed in a symmetric manner: we consider the values i in decreasing order, moving the window from right to left, and keep track of its right part J''_i .

As already explained, by knowing the values f_i and s_i for all $i \in I_k$ and for all k , we immediately get $B[i]$ (and, consequently $LPrF_{g,G}[i]$) for all i .

We now evaluate the running time of our approach. We can compute, in $\mathcal{O}(n)$ time, from the very beginning of our algorithm the partitions of $[1, 2n - 1)$ we need to process (basically, for each I_k we find J_1, J_2 and J_3 in constant time, and we get the three initial partitions we have to process in $\mathcal{O}(\delta)$ time), and we also know the union-operations and find-queries that we will need to perform for each such partition (as we know the order in which the prefixes are taken out of the window, so the order in which the intervals are merged). In total we have $\mathcal{O}(n/\delta)$ partitions, each having initially $\delta + 2$ intervals, and on each we perform δ find-queries and δ -union operations. So, by Remark 1, we can preprocess this data (once, at the beginning of the algorithm) in $\mathcal{O}(n)$ time, to be sure that the time needed to obtain the correct answers to all the find queries is $\mathcal{O}(n)$. So, the total time needed to compute the values f_i for all $i \in I_k$ and for all k is $\mathcal{O}(n)$. Similarly, the total time needed to compute the values s_i for all i is $\mathcal{O}(n)$. Then, for each i we get $B[i]$ and $LPrF_{g,G}[i]$ in $\mathcal{O}(1)$ time.

Therefore, Problem 1(a) can be solved in linear time. □

To solve Problem 1(b) we need to somehow restrict the range where the left arm of the repeat uvu may occur. Thus, we search for u with $2^k \leq |u| \leq 2^{k+1}$, for each $k \leq \log n$; such a factor always start with $w[i + 1..i + 2^k]$ and may only occur in the factor $w[i - G - 2^{k+1}..i - g]$. If $2^k \geq G - g$, using the dictionary of basic factors data structures [15] and the results in [16], we get a constant size representation of the occurrences of $w[i + 1..i + 2^k]$ in that range (which are a constant number of times longer than the searched factor), and then we detect which one of these occurrences produces the repeat uvu with the longest arm. If $2^k < G - g$, we use roughly the same strategy to find the repeat uvu with the longest arm and u starting in a range of radius 2^{k+1} centred around $i - G$ or in a range of length 2^{k+1} ending on $i - g$ (again, these ranges are just a constant

number of times longer than the searched factor). To detect a repeat starting between $i - G + 2^{k+1}$ and $i - g - 2^{k+1}$ we use the strategy from the solution of Problem 1(a); in that case, we just have to return the longest common prefix of $w[i..n]$ and the words $w[j..n]$ with $i - G + 2^{k+1} \leq j \leq i - g - 2^{k+1}$. Overall, this approach can be implemented to work in $\mathcal{O}(n \log n)$ time.

Theorem 2. *Problem 1(b) can be solved in $\mathcal{O}(n \log n)$ time.*

4 Lower Bounded Gap

To solve Problem 2(a) we need to find, for some position i of w , the factor $w[1..j]^R$ with $j \leq i - g(i)$ that occurs closest to $w[1..i]$ in the lexicographically ordered list \mathcal{L} of all the suffixes $w[k..n]$ of w and of the mirror images $w[1..k]^R$ of its prefixes. Doing this for all i takes $\mathcal{O}(n)$ time, as it can be reduced to answering n find queries in an extended interval union-find data structure.

Theorem 3. *Problem 2(a) can be solved in linear time.*

To solve Problem 2(b) we use the following lemma.

Lemma 3. *Given a word w , let $L[i] = \min\{j \mid j < i, LCP(j, i) \geq LCP(k, i) \text{ for all } k < i\}$. The array $L[\cdot]$ can be computed in linear time.*

Another lemma shows how the computation of the array $LPF_g[\cdot]$ can be connected to that of the array $L[\cdot]$. For an easier presentation, let $B[i]$ denote the leftmost starting position of the longest factor x_i that occurs both at position i and at a position j such that $j + |x_i| \leq i - g(i)$; if there is no such factor x_i , then $B[i] = -1$. In other words, the length of the factor x_i occurring at position $B[i]$ gives us $LPF_g[i]$. In fact, $LPF_g[i] = \min\{LCP(B[i], i), i - g(i) - B[i]\}$.

Now, let $L^1[i] = L[i]$ and $L^k[i] = L[L^{k-1}[i]]$, for $k \geq 2$; also, we define $L^+[i] = \{L[i], L[L[i]], L[L[L[i]]], \dots\}$.

The following lemma shows the important fact that $B[i]$ can be obtained just by looking at the values of $L^+[i]$. More precisely, $B[i]$ equals $L^k[i]$, where k is obtained by looking at the values $L^j[i] \leq i - g(i)$ and taking the one such that the factor starting on it and ending on $i - g(i) - 1$ has a maximal common prefix with $w[i..n]$. Afterwards, Theorem 4 shows that this check can be done in linear time for all i , thus solving optimally Problem 2.

Lemma 4. *For a word w of length n and all $1 \leq i \leq n$ such that $B[i] \neq -1$, we have that $B[i] \in L^+[i]$.*

Theorem 4. *Problem 2(b) can be solved in linear time.*

Proof. The main idea in this proof is that, to compute $LPF_g[i]$, it is enough to check the elements $j \in L^+[i]$ with $j \leq i - g(i)$, and choose from them the one for which $\min\{LCP(j, i), i - g(i) - j\}$ is maximum; this maximum will be the value we look for. In the following, we show how we can obtain these values efficiently.

First, if $j = L[i]$ for some i , we define the value $end[j] = k \leq LCP(L[j], i)$, where $|w[j..k]| \leq \min\{LCP(L[j], i), k - L[j]\}$. Basically, for each position $k' \leq k$,

the longest factor x starting at $L[j]$ and ending on k' which also occurs at position i is longer than any factor starting on position j and ending on k' which also occurs at position i . Now we note that if $j \in L^+[i]$ is the greatest element of this set such that $end[j] \leq i - g(i) - 1$, then $LPF_g[i] = \min\{LCP(j, i), i - g(i) - j\}$. Clearly, $end[j]$ can be computed in constant time for each j .

To be able to retrieve efficiently for some i the greatest element of this set such that $end[j] \leq i - g(i) - 1$ we proceed as follows.

First we define a disjoint-set union-find data structure on the universe $U = [1, n + 1)$, where the unions can be only performed between the set containing i and that containing $L[i]$, for all i . Initially, each number between 1 and n is a singleton set in this structure. Moreover, our structure fulfils the conditions that the efficient union-find data structure of [11] should fulfil.

Further, we sort in linear time the numbers $i - g(i)$, for all i ; we also sort in linear time the numbers $end[k]$ for all $k \leq n$. We now traverse the numbers from n to 1, in decreasing order. When we reach position j we check whether j equals $end[k]$ for some k ; if yes, we unite the set containing k with the set containing $end[k]$ for all k such that $end[k] = j$. Then, if $j = i - g(i)$ for some i , we just have to return the minimum of the set containing i ; this value gives exactly the greatest element $j \in L^+[i]$ such that $end[j] \leq i - g(i) - 1$. So, as described above, we can obtain from it the value of $LPF_g[i]$. The computation of this array follows from the previous remarks.

To evaluate the complexity of our approach, note that we do $\mathcal{O}(n)$ union operations and $\mathcal{O}(n)$ find queries on the union-find data structure. By the results in [11], the time needed to construct the union-find data structure and perform these operations on it is also $\mathcal{O}(n)$. From every find query we get in constant time the value of a element $LPF_g[i]$. So the solution of Problem 2 is linear. \square

5 Long Armed Repeats and Palindromes

In this section we solve Problems 3(a) and 3(b).

Recall that a long armed palindrome (respectively, repeat) $w[i..j]vw[i'..j']$ is called maximal if the arms cannot be extended to the right or to the left: neither $w[i..j + 1]v'w[i' - 1..j']$ nor $w[i - 1..j]vw[i'..j' + 1]$ (respectively, neither $w[i..j + 1]v'w[i'..j' + 1]$ nor $w[i - 1..j]v''w[i' - 1..j']$) are long armed palindromes (respectively, repeats). Our solutions rely on the results of [4, 5]. In [4] one proposes an algorithm that, given a word of length n , finds the set S of all its factors which are maximal long armed palindromes in $\mathcal{O}(n + |S|)$ time. In [5] the set of maximal long armed repeats with non-empty gap is shown to be of linear size and is computed in $\mathcal{O}(n)$ time for input words over integer alphabets.

In the following, we essentially show that given the set S of all factors of a word which are maximal long armed palindromes (respectively, repeats) we can compute the array $LPal$ (respectively, $LRep$) for that word in $\mathcal{O}(n + |S|)$ time.

Theorem 5. *Problem 3(a) can be solved in $\mathcal{O}(|S| + n)$, where S is the set of all factors of the input word which are maximal long armed palindromes.*

Proof. We assume that we are given an input word w , for which the set S is computed, using the algorithm from [4].

Let us consider a maximal long armed palindrome $w[i..i+\ell-1]vw[j..j+\ell-1]$, with $w[i..i+\ell-1]^R = w[j..j+\ell-1]$. For simplicity, let us denote by $\delta = |v| = j-i-\ell$, the length of the gap; here, i and $j+\ell-1$ will be called the *outer ends* of this palindrome, while j and $i+\ell-1$ are the *inner ends*.

It is not hard to see that from a maximal palindrome one can get a family of long armed palindromes whose arms cannot be extended by appending letters simultaneously to their outer ends. We now show how this family of long armed palindromes can be computed. Intuitively, we extend simultaneously the gap in both directions, decreasing in this way the length of the arms of the palindrome, until the gap becomes longer than the arm. The longest possible such extension of the gap can be easily computed. Indeed, let $r = \lfloor \frac{\ell-\delta}{3} \rfloor$. It is not hard to check that for $r' \leq r$ we have that $w[i..i+\ell-r'-1]v'w[j+r'..j+\ell-1]$, with $v' = w[i+\ell-r'-1..i+\ell-1]vw[j..j+r'-1]$, is a long armed palindrome whose left arm cannot be extended by appending letters to their outer ends. For $r' > r$ we have that $w[i..i+\ell-r'-1]v'w[j+r'..j+\ell-1]$, with $v' = w[i+\ell-r'-1..i+\ell-1]vw[j..j+r'-1]$, is still a gapped palindrome, but it is not long armed anymore. So, for a maximal long armed palindrome $p = w[i..i+\ell-1]vw[j..j+\ell-1]$, we associate the interval $I_p = [j, j+r]$, and associate to it a weight $g(I_p) = j+\ell-1$. Intuitively, we know that at each position $j' \in I_p$ there exists a factor u , ending at position $j+\ell-1$, such that u^Rv is a suffix of $w[1..j'-1]$ for some v .

On the other hand, if u is the longest factor starting at some position $j' \leq n$ such that u^Rv is a suffix of $w[1..j'-1]$, then the factor $w[i..j] = u^Rvu$ is, in fact, a maximal long armed palindrome x^Ryx (i.e., u^R is a prefix of x^R and u is a suffix of x). In other words, u and u^R could be extended simultaneously inside the gap, but not at the outer ends.

Consequently, to compute $LPal[i]$ for some $i \leq n$ we have to find the long armed palindromes $p \in S$ for which the interval I_p contains i . Then, we identify which of these intervals has the greatest weight. Say, for instance, that the interval I_p in which contains i , is the one that weight maximal weight k from all the intervals containing i . Then $LPal[j] = k - j + 1$. Indeed, from all the factors u starting at position j , such that u^Rv is a suffix of $w[1..j'-1]$ for some v , there is one that ends at position k , while all the other end before k (otherwise, the intervals associated, respectively, to the maximal long armed palindromes containing each of these factors u^Rvu would have a greater weight). So, the palindrome ending at position k is the longest of them all.

This allows us to design the following algorithm for the computation of $LPal[j]$. We first use the algorithm of [4] to compute the set S of all maximal long armed palindromes of w . For each maximal long armed palindrome $p = w[i..i+\ell-1]w[i+\ell..j-1]w[j..j+\ell-1]$, we associate the interval $I_p = [j, j+r]$, where $r = \lfloor \frac{\ell-\delta}{3} \rfloor$ and $\delta = j-i-\ell$, and associate to it the weight $g(I_p) = j+\ell-1$. We process these $|S|$ intervals, with weights and bounds in $[1, n]$, in $\mathcal{O}(n + |S|)$ time as in Lemma 1, to compute for each $j \leq n$ the maximal weight $H[j]$ of an interval containing j . Then we set $LPal[j] = H[j] - j + 1$.

The correctness of the above algorithm follows from the remarks at the beginning of this proof. Its complexity is clearly $\mathcal{O}(|S| + n)$. \square

The solution of Problem 3(b) is very similar. First, we produce the list of all maximal long armed repeats with non-empty gap, using the algorithm from [5]. This takes $\mathcal{O}(n)$ time. Then, just like we did in the previous proof, we compute, for every position i , the prefix u of $w[i..n]$ such that there exists a suffix uv of $w[1..i - 1]$ with $0 < |v| \leq |u|$. Then, for each position i , we compute using Lemma 2 the longest square centred on position i . The answer to our problem is obtained by just taking the maximum of these two values, for each i .

We get the following result.

Theorem 6. *Problem 3(b) can be solved in $\mathcal{O}(n)$.*

References

1. Gusfield, D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, New York (1997)
2. Brodal, G.S., Lyngsø, R.B., Pedersen, C.N.S., Stoye, J.: Finding maximal pairs with bounded gap. In: Crochemore, M., Paterson, M. (eds.) CPM 1999. LNCS, vol. 1645, pp. 134–149. Springer, Heidelberg (1999)
3. Kolpakov, R.M., Kucherov, G.: Finding repeats with fixed gap. In: Proceedings of SPIRE, pp. 162–168 (2000)
4. Kolpakov, R., Kucherov, G.: Searching for gapped palindromes. Theor. Comput. Sci. **410**, 5365–5373 (2009)
5. Kolpakov, R., Podolskiy, M., Posypkin, M., Khrapov, N.: Searching of gapped repeats and subrepetitions in a word. In: Kulikov, A.S., Kuznetsov, S.O., Pevzner, P. (eds.) CPM 2014. LNCS, vol. 8486, pp. 212–221. Springer, Heidelberg (2014)
6. Crochemore, M., Iliopoulos, C.S., Kubica, M., Rytter, W., Waleń, T.: Efficient algorithms for two extensions of LPF table: the power of suffix arrays. In: van Leeuwen, J., Muscholl, A., Peleg, D., Pokorný, J., Rumpe, B. (eds.) SOFSEM 2010. LNCS, vol. 5901, pp. 296–307. Springer, Heidelberg (2010)
7. Crochemore, M., Tischler, G.: Computing longest previous non-overlapping factors. Inf. Process. Lett. **111**, 291–295 (2011)
8. Crochemore, M., Ilie, L., Iliopoulos, C.S., Kubica, M., Rytter, W., Walen, T.: Computing the longest previous factor. Eur. J. Comb. **34**, 15–26 (2013)
9. Kolpakov, R., Kucherov, G.: Finding maximal repetitions in a word in linear time. In: Proceedings of FOCS, pp. 596–604 (1999)
10. Kärkkäinen, J., Sanders, P., Burkhardt, S.: Linear work suffix array construction. J. ACM **53**, 918–936 (2006)
11. Gabow, H.N., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. In: Proceeding of STOC, pp. 246–251 (1983)
12. Duval, J.-P., Kolpakov, R., Kucherov, G., Lecroq, T., Lefebvre, A.: Linear-time computation of local periods. In: Rován, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 388–397. Springer, Heidelberg (2003)
13. Kosaraju, S.R.: Computation of squares in a string (preliminary version). In: Crochemore, M., Gusfield, D. (eds.) CPM 1994. LNCS, vol. 807, pp. 146–150. Springer, Heidelberg (1994)

14. Xu, Z.: A minimal periods algorithm with applications. In: Amir, A., Parida, L. (eds.) CPM 2010. LNCS, vol. 6129, pp. 51–62. Springer, Heidelberg (2010)
15. Crochemore, M., Rytter, W.: Usefulness of the Karp-Miller-Rosenberg algorithm in parallel computations on strings and arrays. *Theoret. Comput. Sci.* **88**, 59–82 (1991)
16. Kociumaka, T., Radoszewski, J., Rytter, W., Waleń, T.: Efficient data structures for the factor periodicity problem. In: Calderón-Benavides, L., González-Caro, C., Chávez, E., Ziviani, N. (eds.) SPIRE 2012. LNCS, vol. 7608, pp. 284–294. Springer, Heidelberg (2012)

Quasiperiodicity and Non-computability in Tilings

Bruno Durand¹ and Andrei Romashchenko^{1,2}(✉)

¹ LIRMM, University of Montpellier and CNRS, Montpellier, France

² IITP RAS, Moscow, Russia

`andrei.romashchenko@lirmm.fr`

Abstract. We study tilings of the plane that combine strong properties of different nature: combinatorial and algorithmic. We prove the existence of a tile set that accepts only quasiperiodic and non-recursive tilings. Our construction is based on the fixed point construction [12]; we improve this general technique and make it enforce the property of local regularity of tilings needed for quasiperiodicity. We prove also a stronger result: any Π_1^0 -class can be recursively transformed into a tile set so that the Turing degrees of the resulting tilings consists exactly of the upper cone based on the Turing degrees of the latter.

Keywords: Tiling · Wang tiles · Computability · Quasiperiodicity · Fixed point

1 Introduction

Tilings form a popular basis for many mathematical games, for games for the kids. In science, they are popular tools for rather different researches, in chemistry (to describe quasicrystalline structures, e.g., [6]), in pure logics (e.g. deciding classes of first order predicates defined on their syntax, see [4]), in computational complexity (as basic model for complexity, [5]). The first famous result about tilings is the so-called domino problem: Berger proved that given a tile set, we cannot decide algorithmically whether it can tile the plane, [1]. Within the proof, Berger constructed the first aperiodic tile set — a tile set that can tile the plane but only non-periodically. It was the first tile set that allows only tilings of the plane with rather complex structure. Thus, rather simple local rules can imply quite nontrivial global structure of a tiling.

Since Berger’s paper, quite a lot of different algorithmic and combinatorial properties of aperiodic tilings were investigated. It was proven that a tile set that accepts only aperiodic tilings, must accept uncountably many of them, [13]. Many researchers tried to construct possibly simpler aperiodic tile sets (e.g., [2, 3, 9, 10, 12]). The idea of “simplicity” was interpreted in several different ways: as the number of tiles, algorithmic simplicity of the construction, etc. Another avenue of research was constructing tile sets that guarantee not only aperiodicity, but also more sophisticated properties of tilings: non-recursive,

maximal algorithmic complexity (of each tiling), robustness and fault-tolerance of tilings, and their combinations, [7, 8, 11, 12].

The fundamental question “*How complex can a tiling be?*” also can be understood in terms of Turing degrees of unsolvability. Some partial answers to this question are known. First of all, we remark that for each tile set, the set of valid tilings is effectively closed (i.e., belongs to the class Π_1^0). In [11] the property of cone-avoidance was proven: for each tile set τ and for every undecidable set A there exists a τ -tiling T such that A is not Turing-reducible to T . Quite a complete study of Turing degrees of tilings was given in [16] and [17].

Not surprisingly, the constructions that guarantee some nontrivial combinatorial properties or involve simulation of a Turing machine require very different technical features. So it is rather difficult to combine in one and the same tiling properties of different nature. In this paper we try to do some kind of aggregation; we combine the combinatorial property of quasiperiodicity with complexity issues. We prove that all upper cones of Turing degrees above any Π_1^0 class can be achieved by a tile set that produces only quasiperiodic tilings. This rather complex theorem has a more concrete consequence: we build a tile set that produces only quasiperiodic tilings, and none of these tilings is recursive.

Let us be more precise now. In this paper *Wang tiles* are unit squares with colored sides. A *tile set* is a finite family of tiles. For a given tile set the domino problem is to decide whether the entire plane can be tiled with these tiles. Here we assume of course that we are given infinitely many copies of each tile (tiles are prototypes); in other words, we are allowed to place translated copies of the same tile into different sites of the plane (rotations are not allowed). In a correct tiling the tiles in the neighbor cells must match (sides in contact must have the same color).

If a tile set τ tiles the plane, we call these tilings τ -tilings. More formally, a τ -tiling can be defined as a mapping $F: \mathbb{Z}^2 \mapsto \tau$, where for each pair of neighboring cells $x, y \in \mathbb{Z}^2$ the colors of the tiles $F(x)$ and $F(y)$ match each other on their neighboring sides. A tiling is called *periodic* if some nontrivial shift transforms it into itself. A tiling F is called *quasiperiodic* (or *uniformly recurrent*) if every pattern that appears in this tiling, appears in every sufficiently large square of F .

The domino problem (existence of a tiling with a given tile set) is algorithmically undecidable, [1]. An interesting and nontrivial fact (which follows from Berger’s theorem) is that there exist tile sets that allow only aperiodic tilings of the plane.

The main result of this article is the following theorem that claims that some tile sets enforce at once two nontrivial properties of a tiling: quasiperiodicity and non-computability.

Theorem 1. *There exists a tile set (a set of Wang tiles) τ such that (i) there exist τ -tilings of the plane, (ii) all τ -tilings are quasiperiodic, (iii) all τ -tilings are non-computable.*

The tile set from Theorem 1 is *not minimal* (we cannot claim that all τ -tilings contain the same finite pattern). In fact, minimality cannot be combined with non computability: each minimal tile set allows at least one computable tiling, see [15]. On the other hand, minimality can be combined with aperiodicity, see Theorem 3 below.

Theorem 2. *For every effectively closed set \mathcal{A} there exists a tile set τ such that (i) all τ -tilings are quasiperiodic, (ii) the Turing degrees of all τ -tiling make up exactly the upper cone of \mathcal{A} (i.e., the class of all Turing degrees d such that $d \geq_T \omega$ for at least one $\omega \in \mathcal{A}$).*

For every tile set τ , the set of τ -tilings is always effectively closed. Moreover, if all τ -tilings are strongly quasiperiodic, then the class of Turing degrees of all τ -tilings is known to be upward closed, see [17]. Thus, Theorem 2 gives a precise characterisation of the Turing spectra of quasiperiodic tilings: they are exactly the upward closed sets in Π_1^0 . Notice that Theorem 2 does not imply the result of [18] (a construction of a minimal subshift of finite type with a nontrivial Turing spectrum that consists of uncountably many cones with disjoint bases). The reason is again that the tile sets we construct are not minimal subshifts (though every tiling for our tile set is uniformly recurrent).

We prove Theorems 1 and 2 using the technique of fixed-point tilings from [12], with some suitable extensions. Though conceptually this technique is not very difficult, in order to meet the space limitations of the conference proceedings and also to make the argument more accessible, we present it in a rather informal way, starting with a proof of a simpler Theorem 3 below. Being somewhat sketchy, we nevertheless do not skip any important part of the construction, and we emphasise the parallels and differences with the previously known construction of a fixed-point tilings in [12].

The rest of the paper is organised as follows. First we remind the reader the core ideas of the fixed-point tiling from [12] and explain how this technique implies aperiodicity of tilings. Then we upgrade the construction and build a tile set that combines the properties of aperiodicity and quasiperiodicity. After that we adapt the proof for the main results of the paper.

2 Self-simulating Tilings (Reminder)

Our proof is based on the *fixed point* construction from [12]. The main idea of this argument is that we can enforce in a tiling a kind of *self-similar* structure. In what follows we remind the principal ingredients of this construction (here we follow the notations from [12]). The reader familiar with this fixed-point technique may skip this section and go directly to Sect. 3.

Let τ be a tile set and $N > 1$ be an integer. We call by a *macro-tile* an $N \times N$ square correctly tiled by matching tiles from τ . Every side of a τ -macro-tile contains a sequence of N colors (of tiles from τ); we refer to this sequence as a *macro-color*. Further, let ρ be some set of τ -macro-tiles (of size $N \times N$). We say that τ *implements* ρ if (i) some τ -tilings exist, and (ii) for every τ -tiling

there exists a unique lattice of vertical and horizontal lines that cuts this tiling into $N \times N$ macro-tiles from ρ . (We do not require that all macro-tiles from ρ appear in every τ -tiling.) The value of N is called the *zoom factor* of this implementation.

A tile set τ is called *self-similar* if it implements some set ρ of τ -macro-tiles with some zoom factor $N > 1$ and ρ is isomorphic to τ . This means that there exist a one-to-one correspondence between τ and ρ such that the matching pairs of τ -tiles correspond exactly to the matching pairs of ρ -macro-tiles. By definition, for a self-similar tile set τ each tiling can be uniquely split into $N \times N$ macro-tiles (the set of all macro-tiles is isomorphic to the initial tile set τ); further, the grid of macro-tiles can be grouped into blocks of size $N^2 \times N^2$, where each block is a macro-tile of rank 2 (again, the set of all macro-tiles of rank 2 is isomorphic to the initial tile set τ), etc. It is not hard to deduce from this observation the following statement.

Proposition 1 (Folklore). *A self-similar tile set τ has only aperiodic tilings.*

The proof is based on a simple observation: every period (if it exists) should be a multiple of N , since the lattice of vertical and horizontal lines that cuts this tiling into $N \times N$ macro-tiles must be *unique*. Similarly, a period must be a multiple of N^2 (to respect the uniquely defined grid of macro-macro-tiles), a multiple of N^3 , etc. It follows that a period must be greater than any integer; see details in [12]. Thus, if we want to construct an aperiodic tile set, then it is enough to present an instance of a self-similar tile set. Below we discuss a very general construction of self-similar tile sets.

2.1 Implementing Some Given Tile Set with a Large Enough Zoom Factor

Assume that we have a tile set ρ where each color is a k -bit string (i.e., the set of colors $C \subset \{0, 1\}^k$) and the set of tiles $\rho \subset C^4$ is presented by a predicate $P(c_1, c_2, c_3, c_4)$ (the predicate is true if and only if the quadruple (c_1, c_2, c_3, c_4) corresponds to a tile from ρ). Assume that we have some Turing machine \mathcal{M} that computes P . Let us show how to implement ρ using some other tile set τ , with a large enough zoom factor N .

We will build a tile set τ where each tile “knows” its coordinates modulo N . This information is included in the tiles’ colors. More precisely, for a tile that is supposed to have coordinates (i, j) modulo N , the colors on the left and on the bottom sides should involve (i, j) , the color on the right side should involve $(i + 1 \pmod N, j)$, and the color on the top side, respectively, involves $(i, j + 1 \pmod N)$, see Fig. 1. This means that every τ -tiling can be uniquely split into blocks (macro-tiles) of size $N \times N$, where the coordinates of cells ranges from $(0, 0)$ in the bottom-left corner to $(N - 1, N - 1)$ in top-right corner, Fig. 2. So, intuitively, each tile “knows” its position in the corresponding macro-tile.

In addition to the coordinates, each tile in τ should have some supplementary information encoded in the colors on its sides. We refer to this additional

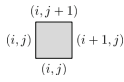


Fig. 1. A tile instance.

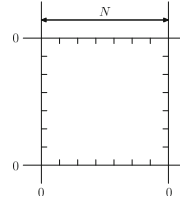


Fig. 2. A macro-tile.

information as the *shade* of the color. On the border of a macro-tile (where one of the coordinates is zero) only two additional shades (say, 0 and 1) are allowed. Thus, for each macro-tile of size $N \times N$ the corresponding macro-colors represent a string of N zeros and ones. We will assume that $k \ll N$. We allocate k bits in the middle of a macro-tile sides and make them represent colors from C ; all other bits on the sides of a macro-tile are zeros.

Now we introduce additional restrictions on tiles in τ that will guarantee the required property: the macro-colors on the macro-tiles satisfy the relation P . To achieve this, we ensure that bits from the macro-tile side are transferred to the central part of the tile, and the central part of a macro-tile is used to simulate a computation of the predicate P . We fix which cells in a macro-tile are “wires” (we may assume that wires do not cross each other) and then require that these tiles carry the same (transferred) bit on two sides. The central part of a macro-tile (of size, say $m \times m$) should represent a time-space diagram of \mathcal{M} ’s computation (the tape is horizontal, time goes up). This is done in a standard way. We require that computation terminates in an accepting state (if not, no correct tiling can be formed), see Fig. 3.

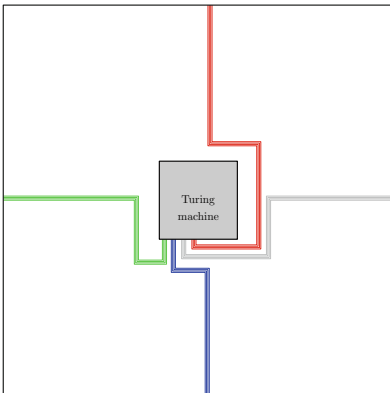


Fig. 3. A macro-tile with an embedded TM.

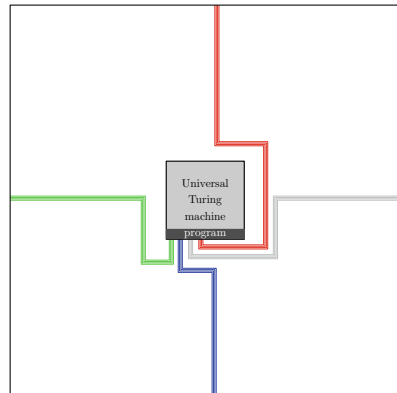


Fig. 4. A macro-tile with an embedded universal TM.

To make this construction work, the size of macro-tile (the number N) should be large enough: first, we need enough space for k bits to propagate, second, we need enough time (i.e., height) so all accepting computations of \mathcal{M} terminate in time m and on space m (where the size of the computation zone m cannot be greater than the size of a macro-tile).

In this construction the number of additional shades depends on the machine \mathcal{M} (the more states it has, the more additional shades we need to simulate the computation in the space-time diagram). To avoid this dependency, we replace \mathcal{M} by a fixed universal Turing machine \mathcal{U} that runs a program simulating \mathcal{M} . We may assume that the tape has an additional read-only layer. Each cell of this layer carries a bit that never changes during the computation; these bits are used as a program for the universal machine. So in the computation zone the columns carry unchanged bits; the construction of a tile set guarantees that these bits form the program for \mathcal{U} , and the computation zone of a macro-tile represents a view of an accepting computation for that program, see Fig. 4. In this way we get a tile set τ that has $O(N^2)$ tiles and implements ρ . (This construction works for all large enough N .)

In the updated construction the tile set still depends on the program simulated in the computational zone. However, this dependency is essentially reduced: the simulated program (and, implicitly, the predicate P) affects only the rules for the tiles used in bottom line of the computational zone. The colors on the sides of all other tiles are universal and do not depend on the simulated tile set τ .

2.2 A Self-similar Tile Set: Implementing Itself

In the previous section we explained how to implement a given tile set ρ (represented as a program for the universal TM) by another tile set τ with large enough zoom factor N . Now we want τ be isomorphic to ρ . This can be done using a construction that follows Kleene’s fixed-point theorem. Note that most steps of the construction of τ do not depend on the program for \mathcal{M} (the coordinates of tiles that make the skeleton of a macro-tile, the information transfer along the wires, the propagation of unchanged program bits, and the space-time diagram for the universal machine in the computation zone). Let us fix these rules as part of ρ ’s definition and set $k = 2 \log N + O(1)$, so that we can encode $O(N^2)$ colors by k bits. From this definition we obtain a program π for the TM that checks that macro-tiles behave like τ -tiles in this respect. We are almost done with the program π . The only remaining part of the rules for τ is the hardwired program. We need to guarantee that the computation zone in each macro-tile carries the very same program π . But since the program is written on the tape of the universal machine, it can be instructed to access its own bits and check that if a macro-tile belongs to the computation zone, this macro-tile carries the correct bit of the program.

It remains to explain the choice of N and m (note that the value of the zoom factor N and the size of the computation zone m are hardwired in the program). We need it to be large enough so the computation described above (which deals

with inputs of size $O(\log N)$) can fit in the computation zone. The computations are rather simple (polynomial in the input size, i.e., $O(\log N)$), so they easily fit in space and time bounded by $m = \text{poly}(\log N)$. This completes the construction of a self-similar aperiodic tile set.

Now it is not hard to verify that the constructed tile sets (1) allows a tiling of the plane, and (2) each tiling is self-similar. Applying Proposition 1 we obtain the following proposition.

Proposition 2 (R. Berger). *There exists a tile set τ such that there exist τ -tilings of the plane, and each τ -tiling is aperiodic.*

In the next section we will upgrade the basic construction of the fixed-point tile set. So far we should keep in mind that in such a tile set all tiles can be classified into three types:

- the “skeleton” tiles that keep no information except for their coordinates in a macro-tile; these tiles work as building blocks for our hierarchical structure;
- the “wires” that transmit the bits of macro-colors from the frontier of the macro-tile to the computation zone;
- the tiles of the computation zone (intended to simulate the space-time diagram of the Universal Turing machine).

The same is true for macro-tiles, super-macro-tiles, etc.; i.e., each macro-tile is a “skeleton” block, or a part of a “wire”, or a cell in the computation zone in the macro-tile of higher rank.

3 Quasiperiodicity and Aperiodicity

Before we approach the main result, we prove a simpler statement; we show that there exists a tile set such that all tilings are both *quasiperiodic* and *aperiodic*.

Theorem 3. *There exists a tile set (a set of Wang tiles) τ such that (i) there exist τ -tilings of the plane; (ii) each τ -tiling is quasiperiodic; moreover, the set of τ -tilings is minimal (i.e., all τ -tilings contain the same finite patterns); (iii) each τ -tiling is aperiodic.*

This result was originally proven in [14] (for a tile set τ constructed in [10]).

3.1 Supplementary Features: What Else We Can Assume on the Fixed-Point Tiling

The general construction of a fixed-point tiling does not imply the property of quasiperiodicity. In fact, for tilings described above, each pattern that includes only “skeleton” tiles (or “skeleton” macro-tiles of some rank k) must appear infinitely often, in all homologous position inside all macro-tiles of higher rank. However, this is not the case for patterns that include tiles from the “communication zone” or the “communication wires”. Informally, the problem is that

even a very small pattern can involve the information relevant for a macro-tile of arbitrarily high rank. So we cannot guarantee that a similar pattern appears somewhere in the neighborhood. To overcome this difficulty we need some new idea and new technical tricks. First of all, without essential modification of the construction we can enforce the following additional properties of a tiling:

- In each macro-tile, the size of the computation zone m is much less than the size of the macro-tile N . Technically, in what follows we will need to reserve free space in a macro-tile to insert $O(1)$ (some constant number) of copies of each 2×2 pattern from the computation zone (of this macro-tile). This requirement is easy to meet. We may assume that the size of a computation zone in a macro-tile of size $N \times N$ is only $m = \text{poly}(\log N)$.
- We require that the tiling inside the computation zone satisfies the property of 2×2 -*determinicity*: if we know all colors on the borderline of a 2×2 -pattern inside of the computation zone (i.e., a tuple of 8 colors), then we can uniquely reconstruct the 4 tiles of this pattern. Again, we do not need any new idea: this requirement is met if we simulate the space-time diagram of a Turing machine in a natural way.
- The communication channels in a macro-tile (the wires that transmit the information from the macro-color on the borderline of this macro-tile to the bottom line of its computation zone) must be isolated from each other. The distance between every two wires must be greater than 2 from each other. That is, each 2×2 -pattern can touch at most one communication wire.

Also we will need a somewhat more essential modification of the construction. We discuss it in the next section.

4 Proof of Theorem 3

To achieve the property of quasiperiodicity, we should guarantee that every finite pattern that appears once in a tiling, must appear in each large enough square. If a tile set τ is self-similar, then in every τ -tiling each finite pattern can be covered by at most 4 macro-tiles (by a 2×2 -pattern) of an appropriate rank. Thus, to prove Theorem 3 it is enough to guarantee that each 2×2 group of macro-tiles (of each rank) that ever appears in a tiling, must appear in each large enough squares in it. This property is not true for the tile set constructed above. As we noticed above, this is obviously true for a 2×2 pattern that involves only skeleton macro-tiles (we can find an identical pattern in the neighboring macro-tile of the appropriate rank); however, this property can be false for patterns that touch the communication wires or the computation zone. To achieve the desired property we need to modify the basic construction. To this end we implement in our construction one new feature.

The New Feature: Notice that for each 2×2 -window that touches the computation zone or the communication wires there exist only a bounded number c of ways to tile them correctly (and make a correct tiling). This constant c depends on the alphabet of the tape and the number of internal states of the Universal

Turing machine. For each possible position of a 2×2 -window in the computation zone or in the communication wires and for each possible filling of this window by tiles, we reserve a special 2×2 -slot in a macro-tile (somewhere far away from the computation zone and from all communication wires) and define the neighbors around this slot in such a way that only these specific 2×2 patterns can patch it. Note that the tiles around this “know” their real coordinates in the bigger macro-tile, while the tiles inside the slot do not (they “believe” to be tiles in the computation zone, though they are in a “slot” outside of it). An example of such a slot is shown in Fig. 5. In Fig. 6 we show how these “slots” are placed in a macro-tile. This simple trick is the sharpest difference between this construction and the fixed-point tilings known before: now some tiles do not “know” their real position in the ambient macro-tile.

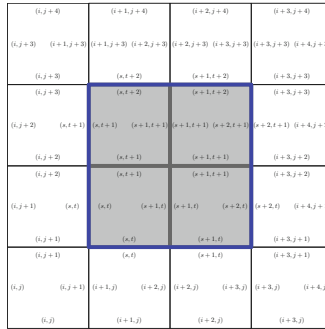


Fig. 5. A ring of 12 “skeleton” tiles (the white squares) makes a slot for a 2×2 -pattern of tiles from the computation zone (the grey squares). In the picture we show the “coordinates” encoded in the colors on the sides of each tile. The “real” coordinates of the bottom-left corner of this slot are $(i + 1, j + 1)$, while the “natural” coordinates of the corresponding patterns (when it appears in the computation zone) are (s, t) (Color figure online).

Here we use (a) the property of 2×2 -determinicity of the computation zone (there is a unique way to put tiles in the “slot”), and (b) the fact that we have enough room to put in a macro-tile the slots for all 2×2 -patters that can appear in the computation zone or along the communication wires. (Here we use the fact that the size of the computational zone $m \times m$ and the lengths of all communication wires $O(1) \times O(N)$ in a macro-tile are much less than the total area of a macro-tile $N \times N$.) This feature guarantees that each 2×2 pattern from the computational zone appears at least once in each macro-tile (such a pattern appears once in each macro-tile in the introduced “slots” and possibly once again in the computation zone of this macro-tile).

We choose the positions of the “slots” in the macro-tile so that coordinates can be computed by a short program in time polynomial in $\log N$. We require that the positions of all slots are disjoint, and they do not touch each other. This precaution is needed to guarantee that the tiles used in the slots do not damage the general structure of the macro-tiles.

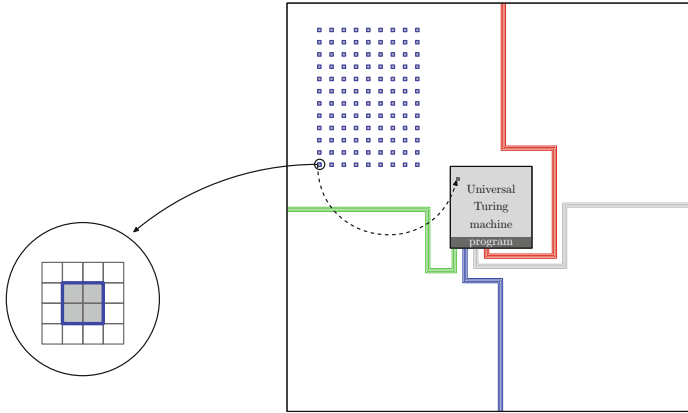


Fig. 6. The array of “slots” (with patterns from the computation zone) embedded in a macro-tile.

For a tile set with a new feature, every tiling enjoys a new property: every 2×2 -pattern of tiles touching the computation zone or a communication wire, must appear at least once in *each* macro-tile (hence, this pattern must appear in each large enough square). Of course, the property of self-similarity implies that similar statements hold for 2×2 -pattern of macro-tiles of each ranks k .

Thus, we proved that every $N^k \times N^k$ pattern that appear in a τ -tiling, must appear in each large enough square in this tiling; moreover, this pattern must appear in each large enough square in *all* τ -tilings. Hence, the constructed tile set satisfies the requirements of Theorem 3.

5 From Aperiodicity to Non-computability

To prove Theorem 1, we need a slightly more sophisticated construction. We need a self-similar tiling with *variable zoom factor*, see [12] for details. In this version of the construction the size of a macro-tile of rank r is equal to $N_r \times N_r$, for some suitable sequence of zooms N_r , $r = 1, 2, \dots$. We may assume that $N_r = Cr$ for some constant C . Now each macro-tile of rank r must “know” its own rank (that is, the binary representation of r is written on the tape of the Turing machine simulated on the computation zone). This information is used by a macro-tile to simulate the next rank macro-tiles properly. The size of the computational zone m_r should also grow as a function of rank r (easily computable from r); again, we may assume that $m_r = \text{poly}(\log N_r)$.

Also we may require that all macro-tiles of rank r contain in their computational zone the prefix (e.g., of length $\lceil \log r \rceil$) of some infinite sequence $X = x_0x_1x_2\dots$. The bits of this prefix are propagated by wires to the neighboring macro-tiles, so all macro-tiles of the same rank contain the same bits $x_0x_1\dots$. The usual self-simulation guarantees that the bits of X embedded into a macro-tile of rank $r + 1$ extends the prefix embedded in a macro-tile of rank r .

Since the size of the computational zone increases as a function of rank r , the entire tiling of the plane involves an infinite sequence of bits X .

The construction becomes interesting if we can enforce some special properties of the embedded sequence X . For example, we can guarantee that it is not computable. Indeed, let us make the machine in the computation zone do some new job: let it enumerate two non-separable enumerable sets (on each level r we run the simulation for the number of steps that fits the computation zone available in a macro-tile of rank r). Then we can require that X is a separator between these two sets, and in each level, the machine verifies that the (partially) enumerated sets are indeed separated by the given prefix of X . Combining all ingredients together, we obtain a tile set τ , which is self-similar in a generalised sense (with a variable zoom factor), with two nontrivial properties: all τ -tilings are non-computable and quasiperiodic. Thus, we proved Theorem 1.

A technical remark: Notice that in this construction we cannot control precisely the sequence X embedded in the tiling (we can specify the two non-separable enumerable sets that are “enumerated” in a tiling, but we cannot define uniquely the separator X between them). Thus, our tile set accepts tilings corresponding to infinitely many sequences X , and it is not minimal. This is not surprising: if an effectively closed subshift contains no computable points, then it cannot be minimal (see, e.g., [19]). In contrast to the proof of Theorem 2, in the construction used in this section we cannot claim that there exist only $O(1)$ ways to fill by a quadruple of macro-tiles of rank k a slot of size $2N_k \times 2N_k$ placed somewhere in a macro-tiles of rank $(k+1)$. Indeed, these macro-tiles involve a prefix of X , and there exist potentially many different sequences X . However, once X is fixed, there rest only a constant number of 2×2 macro-tiles of rank k that fit the given position in the next level macro-tile. This observation allows to reuse the “new feature” from the proof of Theorem 3.

With essentially the same technique we can prove Theorem 2. We employ again the idea of embedding of an infinite sequence X in a tiling. Technically, we require that all macro-tiles of rank k should involve on their computational zone the same finite sequence of $\log k$ bits, which is understood as a prefix of X ; we guarantee that the prefix embedded in macro-tiles of rank $(k+1)$ is compatible with the prefix available to the macro-tiles of rank k . Further, since \mathcal{A} is in Π_1^0 , we can enumerate the (potentially infinite) list of patterns that should not appear in X . On each level, the macro-tiles allocate some part of the available space and time (limited by the size of the computational zone available on this level) to run the enumeration of \mathcal{A} ; every time a new element of \mathcal{A} is enumerated, the algorithm (simulated in the computational zone) verifies that the found forbidden pattern does not appear in the prefix of X accessible to macro-tiles of this level. Since the computational zone in a macro-tile of rank k becomes bigger and bigger as k increases, the enumeration extends longer and longer. Thus, a sequence X can be embedded in an infinite tiling, if and only if this sequence does not contain any forbidden pattern (i.e., this X belongs to \mathcal{A}).

What are the Turing degrees of tilings in the described tile set? For our tile set, every tiling is defined by three infinite parameters: the sequence of bits X

embedded in this tiling, and two sequences of integers σ_h, σ_v that specifies the shifts (the vertical and the horizontal ones) of macro-tiles of each level relative to the origin of the plane. This information is enough to reconstruct the tiling. Indeed, σ_h and σ_v define the hierarchical structure of the macro-tiles: on each level k we should split the macro-tiles of the previous rank into blocks of size $N_k \times N_k$ (k -level macro-tiles), and there are N_k^2 ways to choose the grid of horizontal and vertical lines that define this splitting. And the content of the computational zones of all macro-tile is defined by the prefixes of X . Conversely, given a tiling as an oracle, we can (computably) extract from it the digits of the sequences X , σ_h , and σ_v . It remains to notice that σ_h and σ_v can be absolutely arbitrarily. Thus, the Turing degree of a tiling is the Turing degree of (X, σ_h, σ_v) , which can be arbitrary degree not less than X . That is, the set of degrees of tilings is exactly the upper closure of \mathcal{A} . So we get the statement of Theorem 2.

Acknowledgements. We thank Laurent Bienvenu and Emmanuel Jeandel for many prolific discussions. We are also very grateful to the three anonymous referees for exceptionally detailed and instructive comments.

References

1. Berger, R.: The undecidability of the domino problem. *Mem. Am. Math. Soc.* **66**, 72 (1966)
2. Robinson, R.M.: Undecidability and nonperiodicity for tilings of the plane. *Invent. Math.* **12**, 177–209 (1971)
3. Grünbaum, B., Shephard, G.C.: *Tilings and Patterns*. A Series of Books in the Mathematical Sciences. W.H. Freeman and Company, New York (1989)
4. Börger, E., Grädel, E., Gurevich, Y.: *The Classical Decision Problem*. Springer Science and Business Media, Heidelberg (2001)
5. van Emde Boas, P.: *Dominoes are Forever*. Universiteit van Amsterdam, Mathematisch Instituut, Amsterdam (1983)
6. Levitov, L.S.: Local rules for quasicrystals. *Commun. Math. Phys.* **119**(4), 627–666 (1988)
7. Hanf, W.: Nonrecursive tilings of the plane I. *J. Symbol. Logic* **39**, 283–285 (1974)
8. Myers, D.: Nonrecursive tilings of the plane II. *J. Symbol. Logic* **39**, 286–294 (1974)
9. Culik II, K., Kari, J.: An aperiodic set of Wang cubes. *J. UCS J. Univers. Comput. Sci.* **1**(10), 675–686 (1996)
10. Ollinger, N.: Two-by-two substitution systems and the undecidability of the domino problem. In: Beckmann, A., Dimitracopoulos, C., Löwe, B. (eds.) *CiE 2008*. LNCS, vol. 5028, pp. 476–485. Springer, Heidelberg (2008)
11. Durand, B., Levin, L.A., Shen, A.: Complex tilings. *J. Symbol. Logic* **73**, 593–613 (2008)
12. Durand, B., Romashchenko, A., Shen, A.: Fixed-point tile sets and their applications. *J. Comput. Syst. Sci.* **78**(3), 731–764 (2012)
13. Durand, B.: Tilings and quasiperiodicity. *Theor. Comput. Sci.* **221**(1), 61–75 (1999)
14. Ballier, A.: *Propriétés structurelles, combinatoires et logiques des pavages*. Ph.D. thesis, Marseille, November 2009
15. Ballier, A., Jeandel, E.: Computing (or not) quasi-periodicity functions of tilings. In *Proceedings 2nd Symposium on Cellular Automata (JAC 2010)*, pp. 54–64 (2010)

16. Jeandel, E., Vanier, P.: Π_1^0 sets and tilings. In: Ogihara, M., Tarui, J. (eds.) TAMC 2011. LNCS, vol. 6648, pp. 230–239. Springer, Heidelberg (2011)
17. Jeandel, E., Vanier, P.: Turing degrees of multidimensional SFTs. *Theor. Comput. Sci.* **505**, 81–92 (2013)
18. Hochman, M., Vanier, P.: A note on Turing degree spectra of minimal (2014). [arXiv:1408.6487](https://arxiv.org/abs/1408.6487)
19. Hochman, M.: Upcrossing inequalities for stationary sequences and applications to entropy and complexity. *Ann. Probab.* **37**(6), 2135–2149 (2009)

The Transitivity Problem of Turing Machines

Anahí Gajardo¹, Nicolas Ollinger², and Rodrigo Torres-Avilés¹ (✉)

¹ Departamento de Ingeniería Matemática and Centro de Investigación En Ingeniería Matemática (CI2MA), Universidad de Concepción, Centro de Modelamiento Matemático (CMM), Universidad de Chile, Casilla 160-C, Concepción, Chile
`{anahi,rtorres}@ing-mat.udec.cl`

² INSA Centre Val de Loire, University of Orléans,
LIFO EA 4022, 45067 Orléans, France
`nicolas.ollinger@univ-orleans.fr`

Abstract. A Turing machine is topologically transitive if every partial configuration — that is a state, a head position, plus a finite portion of the tape — can reach any other partial configuration, provided that they are completed into proper configurations. We study topological transitivity in the dynamical system models of Turing machines with moving head, moving tape and for the trace-shift and we prove its undecidability. We further study minimality, the property of every configuration reaching every partial configuration.

Keywords: Reversible computing · Discrete dynamical systems · Symbolic dynamics · Topological dynamics · Computability

1 Introduction

Turing machines [16] provide a simple mechanical model of computation: an agent equipped with a finite internal memory moves over a tape full of symbols which can read and modify it; its movement as well as its interaction with its memory and the tape is deterministically governed by a transition rule. Eventually, this rule may carry the machine to a halting configuration. A computation can be performed by initializing the tape with an input word and decoding the output on the halting tape. The *universality* of the model, asserted by the Church-Turing thesis, the *undecidability* of the *Halting Problem* and its generalizations like *Rice's theorem* provide convenient tools to assert that any kind of system, even “natural” systems like lattice gases dynamics [12], able to perform universal computation has a particular strong form of unpredictability: even if the complete information about the state of the system is available, determining its asymptotic behavior is impossible. On the other hand, qualitative properties of the system can be, and frequently are, undecidable. Even if all the system parameters are given, there is no general procedure to determine whether the

This work has been supported by ECOS Sud – CONICYT project C12E05 and partially supported by FONDECYT#1140684 and Basal PFB03.

system satisfies a given property. This is the case, for example, of dynamics over a piecewise constant vector field in dimension three [1], where the *Reachability Problem*, *i.e.*, the problem of determining whether one region is reachable from another, is undecidable. Proper dynamical properties, as *periodicity*, *sensitivity*, *transitivity* and *limit set*, have also been proved to be undecidable for some classes of systems, in particular in the context of *cellular automata* [7, 8, 11]. The main difficulty to establish such result is that, contrarily to the Turing machine in the context of the Halting Problem, dynamical properties describe the system behavior over the whole set of configurations. The property studied in this paper, the transitivity of a Turing machine, is of the same nature.

The appropriate point of view is to study Turing machines as proper dynamical systems as introduced by Moore [13] and further developed by Kůrka [9] who formalized two associated topologies and establishes several properties for them. Following that trend, several dynamical properties of Turing machines were recently studied: periodicity [2, 3, 8], entropy [6, 14] and equicontinuity [4]. Here we focus on *topological transitivity*: the existence of a point whose orbit passes close to every other point of the space. Moreover, “most” of the points have this characteristic. This gives to the system some kind of *homogeneity* in the sense that some point wise properties become global when transitivity is present. For example, in a transitive system there is a dichotomy between *almost-equicontinuity* and *sensitivity*: either almost every point is stable or every point is sensitive. On the other hand, transitivity is included in the most accepted definitions of “chaos”.

In the context of Turing machines, transitivity imply that every “local configuration” is reachable from every other; but here the meaning of “local” is ambiguous and it depends on the topology that we choose for our model. Kůrka [9] proposes two different topologies for Turing machines, one gives preponderance to the cell contents around the head (Turing Moving Tape model (TMT)) and the other focuses on the cells that surround the position “0” of the tape (Turing Moving Head model (TMH)). In this paper, we consider both models, as well as the column factor of the TMT model: the *trace-shift*.

We establish that transitivity is undecidable in all the three considered models. The proof is performed by reduction from the Reachability Problem, through a technique that consists into *embedding* a machine inside another in such a way the first avoids transitivity of the second when reachability is satisfied. This technique has proved to be very flexible and useful. It was also used in [3] to prove the undecidability of the existence of periodic points in the TMT model. It is also used here to prove the undecidability of another important dynamical property: *minimality*.

2 Definitions

2.1 Turing Machines

A *Turing machine* \mathcal{M} is a triple (Q, Σ, δ) where Q is the finite *set of states*, Σ is the finite *alphabet* and $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{-1, 0, 1\}$ is the partial *transition function* of \mathcal{M} . The machine is *complete* if δ is a total function.

A *configuration* of \mathcal{M} is a triple (s, c, p) where $s \in Q$ is the state of the machine, $c \in \Sigma^{\mathbb{Z}}$ is the content of the *bi-infinite tape* and $p \in \mathbb{Z}$ is the position of the head on the tape. Instead of (s, c, p) , we might use the following convenient notation: $(\dots c_{-1} \cdot c_0 \dots c_p \cdot c_{p+1} \dots)$, where the *dot* before c_0 is there to distinguish the coordinate 0. A configuration (s, c, p) is *halting* if (s, c_p) is a *halting pair*, that is if $\delta(s, c_p)$ is undefined.

A *transition* of the machine transforms a non-halting configuration (s, c, p) into the configuration $(t, c', p + d)$ where $\delta(s, c_p) = (t, a, d)$ and c' is equal to c in every position except for $c'_p = a$. The transition function is denoted by \vdash and its iteration zero or more times by \vdash^* .

A configuration is *periodic* if the machine returns to that same configuration after a finite number of transitions. The machine is *aperiodic* if it has no periodic configuration.

The machine is *surjective* if every configuration can be obtained in one transition from at least one configuration and *injective* if every configuration can be obtained in one transition from at most one configuration, called its preimage. In a complete machine, surjectivity is equivalent to injectivity.

Every injective machine is a *reversible* machine: it can be assigned a reverse. Indeed, an injective machine is characterized by a pair (ρ, μ) , where $\rho : Q \times \Sigma \rightarrow Q \times \Sigma$ is a partial injective function and $\mu : Q \rightarrow \{-1, 0, 1\}$, such that $\delta(s, a) = (t, b, \mu(t))$ where $\rho(s, a) = (t, b)$ for all state s and symbol a . The *reverse machine* \mathcal{M}^{-1} is the reversible Turing machine (Q, Σ, δ^{-1}) where $\delta^{-1}(t, b) = (s, a, -\mu(s))$ for all s, a, t, b such that $\rho(s, a) = (t, b)$. For every non-halting configuration (s, c, p) transformed by \mathcal{M} into the configuration (t, c', p') , the configuration $(t, c', p' - \mu(t))$ is transformed by \mathcal{M}^{-1} into the configuration $(s, c, p - \mu(s))$. A *starting configuration* of \mathcal{M} is a halting configuration of \mathcal{M}^{-1} and a *starting pair* of \mathcal{M} is a halting pair of \mathcal{M}^{-1} . A reversible Turing machine has as many starting pairs as halting pairs.

A *partial configuration* is a configuration (s, c, p) where c is a partial function defined only on a finite and connected portion of the tape. Transitions extend to partial configurations and are defined only when the head is pointing on a defined symbol. A configuration *completes* a partial configuration if they coincide on the intersection of their domains.

Reachability Problem. *Given a Turing machine and a pair of states, decide if, starting from a configuration in the first state, the machine can reach a configuration in the second state after a finite number of transitions.*

The reachability problem is known to be Σ_1^0 -complete even if the input is restricted to aperiodic reversible Turing machines [8] from a starting pair to a halting pair. Moreover, one can fix the alphabet of the machine to be binary.

Transitivity Problem. *Given a Turing machine, decide if, for every pair of partial configurations, one can complete the first partial configuration in a configuration that reaches the second partial configuration after a finite number of transitions.*

Theorem 1. *The transitivity problem is in Π_2^0 and is Π_1^0 -hard.*

Before proving this main result of the paper, we relate it to topological properties of the Turing machine dynamical systems and introduce a last key ingredient: the SMART machine.

2.2 Topological and Symbolic Dynamics

A *topological dynamical system* is a pair (X, T) , where the topological space X is the *phase space* and the continuous function $T : X \rightarrow X$ is the *global transition function* of the system. The *orbit* of a point $x \in X$ is the infinite sequence $\mathcal{O}(x) = (T^n(x))_{n \in \mathbb{N}}$. A point x is *periodic* if $T^n(x) = x$ for some $n > 0$.

A system (Y, T) is a *subsystem* of (X, T) if $Y \subseteq X$ is closed and $T(Y) \subseteq Y$. The smallest subsystem that contains a given point x is the closure of its orbit $\overline{\mathcal{O}(x)}$. A system (X, T) is *transitive* if it admits a *transitive point*, i.e. a point x such that $\overline{\mathcal{O}(x)} = X$. A system where every point is transitive is a *minimal* system. A system (Z, F) is a *factor* of (X, T) if there exists a continuous and onto function $\varphi : X \rightarrow Z$ such that $F \circ \varphi = \varphi \circ T$. Factors inherit several properties including transitivity. A convenient characterization of transitivity is the following (see [10] for this and other details about dynamical systems).

Proposition 1. *Given a compact and perfect set X , the dynamical system (X, T) is transitive if and only if for every pair of open sets U, V , there exists a time t such that $T^t(U) \cap V \neq \emptyset$.*

Let Σ be a finite alphabet. Words, infinite words and bi-infinite words are respectively finite, right-infinite and bi-infinite sequences of symbols from Σ whose sets are denoted respectively by Σ^* , the *one-sided full shift* $\Sigma^{\mathbb{N}}$ and the *two-sided full shift* $\Sigma^{\mathbb{Z}}$. A finite word v is a *factor* of another (finite or infinite) word z , denoted by $v \sqsubseteq z$, if there exist two indexes i and j , such that $v = z_i z_{i+1} \dots z_j$. The length of a finite word u is denoted by $|u|$.

Let \mathbb{T} denote either \mathbb{Z} or \mathbb{N} . An element x of $\Sigma^{\mathbb{T}}$ is an ordered sequence $x = (x_i)_{i \in \mathbb{T}}$. The *shift* function σ is defined on $\Sigma^{\mathbb{T}}$ by $\sigma(y)_i = y_{i+1}$, and it is a bijective function if $\mathbb{T} = \mathbb{Z}$. The metric of the full shift is the Cantor metric: $d(x, y) = 2^{-i}$, where $i = \min\{|n| : x_n \neq y_n\}$. With this metric, $\Sigma^{\mathbb{T}}$ is compact and $(\Sigma^{\mathbb{T}}, \sigma)$ is a topological dynamical system. The subsystems of $(\Sigma^{\mathbb{T}}, \sigma)$ are called *subshifts*. Frequently, we will prefer to denote $\Sigma^{\mathbb{N}}$ by Σ^ω , the set of right-infinite sequences; and symmetrically, we will use ${}^\omega\Sigma$ to denote the left-infinite sequences.

The *language* $\mathcal{L}(S)$ of a subshift S is the set of its factors, i.e. $\mathcal{L}(S) = \{u \in \Sigma^* \mid \exists z \in S, u \sqsubseteq z\}$. Conversely, every language L defines a subshift $\mathcal{S}_L = \{z \in \Sigma^{\mathbb{N}} \mid \forall u \sqsubseteq z, u \in L\}$. A set S is a subshift if and only if $\mathcal{S}_{\mathcal{L}(S)} = S$.

2.3 Turing Machines Seen as Dynamical Systems

Let $X = Q \times \Sigma^{\mathbb{Z}} \times \mathbb{Z}$ be the set of configurations of a complete Turing machine. Endowing X with the product topology defines a topological dynamical system (X, T) . However, X is not a compact set. Following Kůrka [9], we reformulate X to overcome this problem.

Turing Machine with Moving Head (TMH). In this model, the head is added as an element of the tape; then, the phase space is the set $X_h \subset (\Sigma \cup Q)^{\mathbb{Z}}$, defined by $X_h = \{x \in (\Sigma \cup Q)^{\mathbb{Z}} \mid |\{i \in \mathbb{Z} : x_i \in Q\}| \leq 1\}$. The transition function T_h consists in one application of the local transition function δ , taking in consideration that the head position is at the right of the unique cell that contains a state on the tape. Configurations with no state in the tape are *headless* configurations and are fixed points. The coding function $\psi : X \rightarrow X_h$ transforms a configuration (s, c, p) into $\psi(s, c, p) = x$, where x is defined by $x_i = c_i$ if $i < p$, $x_p = s$ and $x_i = c_{i-1}$ if $i > p$.

By construction, X_h is a subshift. With the Cantor metric, ψ is continuous and one-to-one and $X_h = \psi(X)$. Configurations from $X_h \setminus \psi(X)$ are exactly the headless configurations. The transition function T_h is continuous and partial configurations are represented by factors $x_i \dots x_j$.

Turing Machine with Moving Tape (TMT). In this model, the head is fixed at the origin and it is the tape which moves. The phase space is $X_t = {}^\omega \Sigma \times Q \times \Sigma^\omega$ and T_t consists in one application of δ by moving the tape instead of the head.

The onto coding function $\Gamma : X \rightarrow X_t$ transforms a configuration (s, c, p) into $(\dots c_{p-2}c_{p-1}, s, c_p c_{p+1} \dots)$. Endowing X_t with the product topology results in a compact phase space and both Γ and T_t are continuous. By applying Γ to a configuration, one loses information about the original head position. Partial configurations are triples $(u, s, v) \in \Sigma^* \times Q \times \Sigma^*$.

The Trace-Shift. The *trace-shift* S_t is the column shift associated to the moving tape model. It is obtained from the projection $\pi : X_t \rightarrow Q \times \Sigma$ defined by $\pi(u, s, av) = (s, a)$. The *trace-shift* is the image of the factor map $\tau : X_t \rightarrow S_t$, defined as $\tau(x) = (\pi(T_t^n(x)))_{n \in \mathbb{N}}$. The definition is generalized to partial configurations. The map τ is not invertible, but given a semi-infinite word $w \in S_t$, its pre-image x is uniquely defined over the set of visited cells. For a given word w either in S_t or in $\mathcal{L}(S_t)$, we define its canonical pre-image (u, s, v) as the smallest finite (or infinite) configuration whose image by τ is w .

It is noteworthy to remark that when T is a reversible machine, S_t can be defined as a subshift of $\Sigma^{\mathbb{Z}}$ by redefining $\tau(u, r, u') = (\pi(T_t^n(x)))_{n \in \mathbb{Z}}$, and all of the results that we will establish in this paper remain true.

3 Transitivity of Turing Machines

3.1 Characterizing Transitivity Properties

By proposition 1, the property described in the Transitivity Problem is indeed the topological transitivity in the TMH model.

Proposition 2. *Let (Q, Σ, δ) be a complete Turing machine.*

The TMH system (X_h, T_h) is transitive if and only if for every pair of partial configurations $(u.u')$ and $(v.v')$, there exists a completion $x \in X_h$ of $(u.u')$ and a time $n \in \mathbb{N}$ such that $T_h^n(x)$ is a completion of $(v.v')$.

The TMT system (X_t, T_t) is transitive if and only if for every pair of partial configurations (u, s, u') and (v, t, v') , there exists a completion $(w, s, w') \in X_t$ of (u, s, u') and a time $n \in \mathbb{N}$ such that $T_t^n(w, s, w')$ is a completion of (v, t, v') .

The trace-shift (S_t, σ) is transitive if and only if for every $u, v \in \mathcal{L}(S_t)$, there exists a third word $w \in \mathcal{L}(S_t)$ such that $uwv \in \mathcal{L}(S_t)$.

We see from this that transitivity is in Π_2^0 , since the existing configuration needs to be specified only on a finite number of cells. A peculiarity of Turing models is the relation between transitivity and periodic points of T_h . In these points, the head is enclosed in a finite part of the tape. Any perturbation of the configuration that does not affect this part of the tape will not perturb the head. Thus, no periodic point can be attained by a point outside its orbit, and the system cannot be transitive. Moreover, when T_h has a periodic point, transitivity is excluded both from (X_t, T_t) and (S_t, σ) [5].

Proposition 3. (X_h, T_h) transitive $\stackrel{(1)}{\Rightarrow} (X_t, T_t)$ transitive $\stackrel{(2)}{\Rightarrow} (S_t, \sigma)$ transitive.

Proof. (1) Any finite configuration of X_t corresponds to several finite configurations of X_h , thus if a point exists that visits any finite configuration of X_h , the same point will visit any finite configuration of X_t . (2) (S_t, σ) is a factor of (X_t, T_t) thus it inherits its transitivity. ■

Surjectivity is a necessary condition for transitivity, thus if the TMT or the TMH models are transitive, the machine needs to be reversible. Note this is not the case for trace-shift, as there exist non surjective Turing machines with surjective trace-shift [15].

3.2 The SMART Machine

The SMART machine is the 4-state 3-symbols reversible complete Turing machine depicted on Fig. 1. Among other properties, it is aperiodic, its trace-shift is substitutive and it is minimal in TMT, as proven in [3]. We prove below that it is also transitive in TMH.

The head of SMART zigzags over the lagoons of 0s, either to the right or to the left, depending on its states and the surrounding symbols, as formalized below and proved by recurrence over n in [3].

Lemma 1. For all $n \in \mathbb{N}$ and all symbols $s_+ \in \{1, 2\}$ and $s_* \in \{0, 1, 2\}$,

$$\begin{pmatrix} s_* & 0^n & 0 & s_+ \\ & & b & \end{pmatrix} \vdash^* \begin{pmatrix} s_* & 0^{n+1} & s_+ \\ & & b \end{pmatrix} \tag{B(n)}$$

$$\begin{pmatrix} s_+ & 0 & 0^n & s_* \\ & & d & \end{pmatrix} \vdash^* \begin{pmatrix} s_+ & 0^{n+1} & s_* \\ & & d \end{pmatrix} \tag{D(n)}$$

$$\begin{pmatrix} 0 & 0^n & s_+ \\ & & p \end{pmatrix} \vdash^* \begin{pmatrix} 0^{n+1} & s_+ \\ & p \end{pmatrix} \tag{P(n)}$$

$$\begin{pmatrix} s_+ & 0^n & 0 \\ & & q \end{pmatrix} \vdash^* \begin{pmatrix} s_+ & 0^{n+1} \\ & q \end{pmatrix} \tag{Q(n)}$$

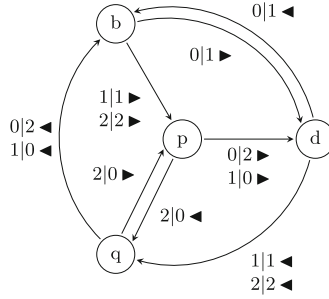


Fig. 1. The SMART machine.

A key lemma for the minimality of SMART in [3] is that every partial configuration can be produced inside a large enough block of 0s guarded by 2s.

Lemma 2. For every word $u \in \{0, 1, 2\}^*$ of length n , state s and every $1 \leq i \leq n$, there exist $k, k' \in \mathbb{N}$ such that $\left(\begin{smallmatrix} .2 & 0^{k+k'+n-3} & 0 & 2 \end{smallmatrix} \right)_b \vdash^* \left(\begin{smallmatrix} .2^k & u_1 & \dots & u_i & \dots & u_n & 2^{k'} \end{smallmatrix} \right)_s$.

We now introduce two new technical lemmas.

Lemma 3. The configuration $\left(\begin{smallmatrix} w & 2 & .2 & 2^w \end{smallmatrix} \right)_p$ reaches each of the configurations of the family $\left\{ \left(\begin{smallmatrix} w & 2 & 0 & 0^k & .0^k & 0 & 2 & 0 & 2^w \end{smallmatrix} \right) \right\}_{k \in \mathbb{N}} \cup \left\{ \left(\begin{smallmatrix} w & 2 & 0 & 0 & 0^k & .0^k & 0 & 2 & 0 & 2^w \end{smallmatrix} \right) \right\}_{k \in \mathbb{N}}$.

Proof. By a simple recurrence over k . From k to $k + 1$ apply the recurrence hypothesis then $B(2k)$, one step, $P(2k)$, one step. From there one step gives the even case. For the odd case apply $Q(2k)$, one step, $P(2k + 2)$ then 2 steps. ■

Lemma 4. For every $k \leq n - 1$, $\left(\begin{smallmatrix} .2 & 0^{n+2} & 0 & 2 \end{smallmatrix} \right)_b$ reaches both $\left(\begin{smallmatrix} .2 & 2 & 0^k & 0 & 2 & 0^{n-k} & 2 \end{smallmatrix} \right)_b$ and $\left(\begin{smallmatrix} .2 & 0^{n-k} & 2 & 0^k & 0 & 2 & 2 \end{smallmatrix} \right)_b$.

Proof. Apply $B(n + 2)$, 2 steps, then $D(n + 1)$. In the first case continue by 2 steps then repeat $n - k - 1$ times the sequence $B(n - i)$, one step, $P(n - i)$, 2 steps, from $i = 0$ to $n - k - 1$. In the second case continue by one step then repeat $n - k - 1$ times the sequence $Q(n - i + 1)$, 2 steps, $D(n - i)$, one step, and finish by one step. ■

Theorem 2. The SMART machine is topologically transitive in TMH.

Proof. By Lemma 2 any possible partial configuration is reachable from a partial configuration x' with a certain amount of 0 in a certain position. Lemma 4 establishes that x' is reachable from a configuration x'' with the 0s in the center. Finally, Lemma 3 asserts that x'' is always reachable from $\left(\begin{smallmatrix} w & 2 & .2 & 2^w \end{smallmatrix} \right)_p$. Therefore, configuration $\left(\begin{smallmatrix} w & 2 & .2 & 2^w \end{smallmatrix} \right)_p$ is a transitive point, and SMART is transitive. ■

4 The Complexity of Topological Transitivity

4.1 Construction Techniques

Our proof combines partial Turing machines to construct bigger ones. One key technique from [8] is *Reversing the time*: given a reversible Turing machine $M = (Q, \Sigma, \delta)$, one creates two new reversible machines $M_+ = (Q \times \{+\}, \Sigma, \delta^+)$, and $M_- = (Q \times \{-\}, \Sigma, \delta^-)$, where $(s, +)$ and $(s, -)$ states represent M in state s running respectively forwards and backwards in time.

The second key technique is the *Embedding* that inserts a machine inside the transitions of another in such a way that the new machine has one or more properties that depends on some properties of the original machines. We distinguish a *host* machine $H = (Q, \Sigma, \delta)$ and a reversible and *innocuous* invited machine I that share the same alphabet.

Definition 1. A reversible machine is innocuous if every starting pair (s, a) is associated to a unique halting pair (t, a) so that the evolution of every starting configuration $(s, c, p + \mu(s))$ where $c(p) = a$ either is infinite or stops in the halting configuration (t, c, p) .

Remark 1. Innocuous machines can be obtained by gluing together the halting pairs of M_+ to the starting pairs of M_- , so that a halting configuration computes back to the starting configuration.

The embedding H^I of the invited machine I in the host machine H is constructed from the disjoint union of H and I . Let $(s_1, a_1), \dots, (s_n, a_n)$ be the starting pairs of I and $(t_1, a_1), \dots, (t_n, a_n)$ be the associated halting pairs. Let $\delta(r, a) = (q, b, \Delta)$ be a fixed transition of H . That transition is removed from H^I and replaced by the following transitions: $\delta(r, a) = (s_1, a_1, \mu(s_1))$, $\delta(t_1, a_1) = (s_2, a_2, \mu(s_2))$, $\delta(t_2, a_2) = (s_3, a_3, \mu(s_3))$, \dots , $\delta(t_n, a_n) = (q, b, \Delta)$, as depicted on Fig. 2. Notice that H^I is complete when H is complete.

If we start at a state of H in the resulting machine, we will see the evolution of H , alternated with some intervals of time in which it is the machine I that evolves.

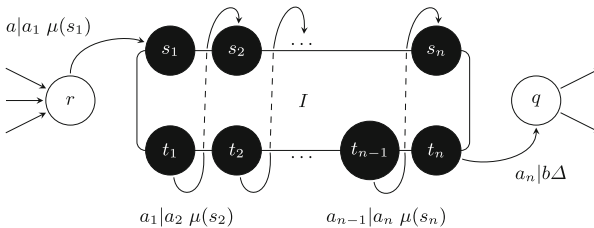


Fig. 2. Embedding technique.

4.2 Undecidability of Transitivities

Theorem 3. *The problem to decide if a given reversible complete Turing machine is transitive is Π^0_1 -hard in all three models: TMH, TMT and trace-shift.*

Proof. The proof proceeds by reduction of the Reachability Problem for binary reversible aperiodic Turing machines. Let M be such a machine with a starting pair (s, a) and a halting pair (t, b) . Let $\$$ be a new symbol not in Σ and consider M' the copy of M with this new symbol: all the pairs $(r, \$)$ are both starting and halting pairs of M' . Let $(p_1, a_1), \dots, (p_m, a_m)$ be all the starting pairs of M' except (s, a) and $(q_1, b_1), \dots, (q_m, b_m)$ be all the halting pairs of M' except (t, b) . Take two copies of M' and apply *Reversing the time* to obtain 4 machines $M'_{1-}, M'_{1+}, M'_{2-}$ and M'_{2+} , then connect them according to Fig. 3 by adding the following transitions to obtain the invited machine I :

$$\begin{aligned} \delta(t^{1+}, b) &= (t^{1-}, b, -\mu(t)), \delta(s^{1-}, a) = (s^{1+}, a, \mu(s)), \\ \delta(t^{2+}, b) &= (t^{2-}, b, -\mu(t)), \delta(s^{2-}, a) = (s^{2+}, a, \mu(s)), \\ \delta(q_i^{1+}, b_i) &= (q_i^{2-}, b_i, -\mu(q_i)) \quad \forall i \in \{1, \dots, m\} \\ \delta(p_i^{1-}, a_i) &= (p_i^{2+}, a_i, \mu(p_i)) \quad \forall i \in \{1, \dots, m\} \end{aligned}$$

The starting pairs of I are the pairs (p_i^{1+}, a_i) and the pairs (q_i^{1-}, b_i) for all i . The machine I is innocuous as only three scenarios are possible from a starting pair: (1) entering M_{1+} (or M_{1-}) and staying there forever; (2) exit M_{1+} by (t^{1+}, b) (or M_{1-} by (s^{1-}, a)), the computation is then reversed inside M_{1-} (or M_{1+}) and enters M_{2+} (or M_{2-}) replaying the same scenario to exit M_{2-} (or M_{2+}) leaving the tape identical to the beginning ; (3) exit M_{1+} (or M_{1-}) by a halting pair (q_i^{1+}, b_i) , the computation is then reversed in M_{2-} (or M_{2+}) leaving the tape identical to the beginning. Consider the embedding SMART^I .

(Assertion 1). SMART^I may be transitive only if M cannot reach (t, b) from (s, a) . Indeed, if (s, a) can reach (t, b) in M then SMART^I admits a periodic point and its trace shift cannot be transitive and by Proposition 3 none of the models can be transitive.

(Assertion 2). The TMH system of SMART^I is transitive if M cannot reach (t, b) from (s, a) . Indeed, suppose that M cannot reach (t, b) from (s, a) , then I is aperiodic, and let (s_u, u, i) and (s_v, v, j) be two partial configurations.

Case 1. s_u and s_v are states of SMART. In this case we know that there exists a finite context (s_u, u', i) that extends (s_u, u, i) so that SMART reaches (s_v, v, j) , because SMART is transitive. Let us complete u' with the symbol $\$$ and lets analyze the behavior of SMART^I over $x = (s_u, {}^\omega\$u'{}^\omega, i)$. First of all, SMART^I cannot stay an infinite amount of time inside I , because I is aperiodic, and non periodic behavior needs an infinite amount of space to be performed. The presence of the extraneous symbol $\$$ in x , avoid this to happen. Now, since I is innocuous, we will see the machine SMART evolving, and thus configuration (s_v, v, j) will be reached.

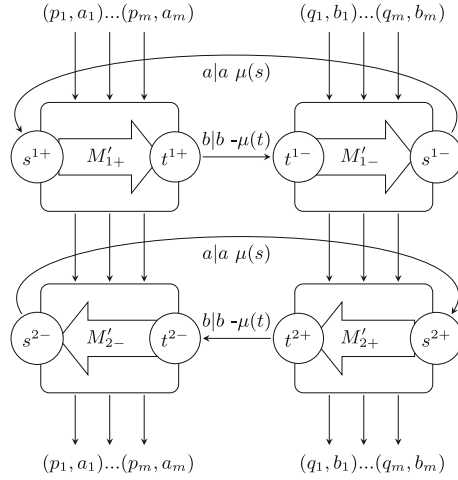


Fig. 3. Invited machine for an embedding that is transitive if and only if (s, a) cannot reach (t, b) in the evolution of M , used in the proof of Theorem 3.

Case 2. s_u or s_v is a state of I . In this case, let's add $\$$ symbols around u and v , and let's evolve the time backward from $(s_v, \$v\$, j)$ until the embedded machine exits the invited machine I , and let's call $(s'_v, \$v'\$, j')$ the obtained finite configuration. In the same way, we evolve the time forward from $(s_u, \$u\$, i)$ until exiting I , and we call $(s'_u, \$u'\$, i')$ the so obtained configuration. If either s_u or s_v are already in SMART, we just add the $\$$ symbols. Now we can apply Case 1 to $(s'_u, \$u'\$, i')$ and $(s'_v, \$v'\$, j')$ to prove the existence of a completion u'' of $\$u'\$$ such that $(s'_u, u'', i') \vdash^* (s'_v, \$v'\$, j')$. Let's suppose that $u'' = w\$u'\w' . Therefore, by construction, we have that $(s_u, w\$u'\$w', i) \vdash^* (s'_u, w\$u'\$w', i') \vdash^* (s'_v, \$v'\$, j') \vdash^* (s_v, \$v\$, j)$, which is the desired conclusion.

From Proposition 3 we know that the classes of transitive machines for TMH, TMT and the trace-shift are nested, Assertions 1 and 2 prove that all three related problems are Π_1^0 -hard. ■

5 The Complexity of Minimality

A dynamical system (X, T) is *minimal* if $\overline{\mathcal{O}(x)} = X$ for all $x \in X$. It is equivalent to not have any no trivial proper subsystem. Every minimal system is also transitive. In the context of Turing machines, we will find machines which have a minimal TMT and a minimal trace-shift, the SMART machine is an example of this [3]. There is no machine with a minimal TMH system, because this system always contains fixed points: the headless configurations. As for transitivity, minimality in the TMT model imply minimality in the trace-shift, by the factor relation.

Theorem 4. *The problem to decide if a given reversible complete Turing machine is minimal is Σ_1^0 -hard for both TMT and trace-shift.*

Proof. The proof proceeds by reduction of the *Mortality problem* for reversible and aperiodic machines, proved undecidable in [8]. A Turing machine is mortal if every configuration eventually halts. The mortality problem, that is to decide, given a Turing machine, if it is mortal, is Σ_1^0 -complete for reversible Turing machines.

Let M be a reversible and aperiodic ternary Turing machine. Apply *Reversing the time* to generate two machines M_+ and M_- and combine them into an invited machine as per Fig. 4: for every halting pair (q_i, b_i) of M , add a transition $\delta(q_i^+, b_i) = (q_i^-, b_i, -\mu(q_i))$. The machine I is innocuous, and we embed it into SMART to produce SMART^I .

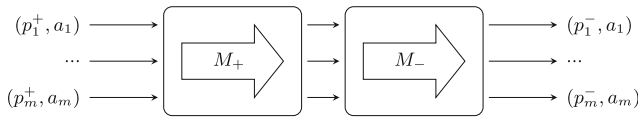


Fig. 4. Invited machine for an embedding that is minimal if and only if M is mortal, used in the proof of Theorem 4.

(*Assertion 1*). If the trace-shift of SMART^I is minimal, then M needs to be mortal. Indeed, if M is not mortal, there is a trace that starts in a state of M_+ and never exits this machine. Such a behavior is impossible in a machine with a minimal trace-shift, where all the trajectories need to be transitive, and so they must visit all the states of the machine.

(*Assertion 2*). If M is mortal, the TMT system of SMART^I is minimal. Indeed, let's suppose that M is mortal, so its reverse is mortal too, and so is I . Let x be an arbitrary configuration in the TMT system of SMART^I , we will prove that it reaches every finite configuration (v, r, v') in the TMT system. First, if r is a state of SMART and x has also a state of SMART, it is clear that x reaches (v, r, v') because SMART is minimal and it is impossible to stay an infinite amount of time inside I . Second, if x has a state in I , it comes from a configuration x' that do has a state in SMART, because the inverse of I is mortal. The orbit of x is equal to the orbit of x' except for a finite number of points, whose state is in I , thus, by the previous argument, x can attain any finite configuration with a state in SMART. Now, if r is a state of I , we can evolve SMART^I backward on (v, r, v') until to arrive to configuration (u, r', u') with a state r' of SMART, this is always possible because I is mortal. Through the former arguments, we know that x visits (u, r', u') , it thus visits (v, r, v') too, and we conclude the proof of assertion 2.

The class of machines with a minimal TMT system is contained in the class of machines with a minimal trace-shift, thus Assertion 1 and 2 prove that the two related problems are Σ_1^0 -hard. ■

References

1. Asarin, E., Maler, O., Pnueli, A.: Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theor. Comput. Sci.* **138**(1), 35–65 (1995)
2. Blondel, V.D., Cassaigne, J., Nichitiu, C.: On the presence of periodic configurations in Turing machines and in counter machines. *Theor. Comput. Sci.* **289**, 573–590 (2002)
3. Cassaigne, J., Ollinger, N., Torres-Avilés, R.: A Small Minimal Aperiodic Reversible Turing Machine (2014, submitted to a journal). <https://hal.archives-ouvertes.fr/hal-00975244v1>
4. Gajardo, A., Guillon, P.: Zigzags in turing machines. In: Ablayev, F., Mayr, E.W. (eds.) CSR 2010. LNCS, vol. 6072, pp. 109–119. Springer, Heidelberg (2010)
5. Gajardo, A., Mazoyer, J.: One head machines from a symbolic approach. *Theor. Comput. Sci.* **370**, 34–47 (2007)
6. Jeandel, E.: Computability of the entropy of one-tape Turing machines. In: Mayr, E., Portier, N. (eds.) Symposium on Theoretical Aspects of Computer Science (STACS 2014), vol. 25, pp. 421–432 (2014)
7. Kari, J.: Rice’s theorem for the limit sets of cellular automata. *Theor. Comput. Sci.* **127**(2), 229–254 (1994)
8. Kari, J., Ollinger, N.: Periodicity and immortality in reversible computing. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 419–430. Springer, Heidelberg (2008)
9. Kůrka, P.: On topological dynamics of Turing machines. *Theor. Comput. Sci.* **174**(1–2), 203–216 (1997)
10. Kůrka, P.: Topological and Symbolic Dynamics. Société Mathématique de France, Paris (2003)
11. Lukkarila, V.: Sensitivity and topological mixing are undecidable for reversible one-dimensional cellular automata. *Cell. Automata* **5**(3), 241–272 (2010)
12. Margolus, N.: Physics and computation. Ph.D. thesis, M.I.T., Cambridge, Mass., U.S.A. (1987)
13. Moore, C.: Generalized shifts: unpredictability and undecidability in dynamical systems. *Nonlinearity* **4**(2), 199–230 (1991)
14. Oprocha, P.: On entropy and turing machine with moving tape dynamical model. *Nonlinearity* **19**, 2475–2487 (2006)
15. Torres, R., Ollinger, N., Gajardo, A.: Undecidability of the surjectivity of the subshift associated to a turing machine. In: Glück, R., Yokoyama, T. (eds.) RC 2012. LNCS, vol. 7581, pp. 44–56. Springer, Heidelberg (2013)
16. Turing, A.: On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc.* **42**(2), 230–265 (1936)

Strong Inapproximability of the Shortest Reset Word

Paweł Gawrychowski¹ and Damian Straszak² (✉)

¹ Institute of Informatics, University of Warsaw, Warsaw, Poland
gawry@mimuw.edu.pl

² EPFL, Lausanne, Switzerland
damain.straszak@epfl.ch

Abstract. The Černý conjecture states that every n -state synchronizing automaton has a reset word of length at most $(n - 1)^2$. We study the hardness of finding short reset words. It is known that the exact version of the problem, i.e., finding the shortest reset word, is NP-hard and coNP-hard, and complete for the DP class, and that approximating the length of the shortest reset word within a factor of $O(\log n)$ is NP-hard [Gerbush and Heeringa, CIAA'10], even for the binary alphabet [Berlinkov, DLT'13]. We significantly improve on these results by showing that, for every $\varepsilon > 0$, it is NP-hard to approximate the length of the shortest reset word within a factor of $n^{1-\varepsilon}$. This is essentially tight since a simple $O(n)$ -approximation algorithm exists.

1 Introduction

Let $A = (Q, \Sigma, \delta)$ be a deterministic finite automaton. We say that $w \in \Sigma^*$ resets (or synchronizes) A if $|\delta(Q, w)| = 1$, meaning that the state of A after reading w does not depend on the choice of the starting state. If at least one such w exists, A is called synchronizing. In 1964 Černý conjectured that every synchronizing n -state automaton admits a reset word of length $(n - 1)^2$. The problem remains open as of today. It is known that an $\frac{n^3-n}{6}$ bound holds [16] and that there are automata requiring words of length $(n - 1)^2$. The conjecture was proved for various special classes of automata [1, 8, 11, 14, 17, 18]. For a thorough discussion of the Černý conjecture see [19].

Computational problems related to synchronizing automata were also studied. It is known that finding the shortest reset word is both NP-hard and coNP-hard [8]. Moreover, it was shown to be DP-complete [15].

In this paper, rather than looking at the exact version, we consider the problem of finding short reset words for automata, or to put it differently, the question

Supported by the *NCN* grant 2011/01/D/ST6/07164.

P. Gawrychowski—Currently holding a post-doctoral position at Warsaw Center of Mathematics and Computer Science.

D. Straszak—Part of the work was carried out while the author was a student at Institute of Computer Science, University of Wrocław, Poland.

of approximating the length of the shortest reset word. For a given n -state synchronizing automaton, we want to find a reset word which is at most α times longer than the shortest one, where α can be either a constant or a function of n . There is a simple polynomial time algorithm achieving $O(n)$ -approximation [10].

Previous Work and Our Results. Berlinkov showed that finding an $O(1)$ -approximation is NP-hard by giving a combinatorial reduction from SAT [6]. Later, Gerbush and Heeringa [10] used the $\log n$ -approximation hardness of SetCover [9] to prove that $O(\log n)$ -approximation of the shortest reset word is NP-hard. Finally, Berlinkov [7] extended their result to hold even for the binary alphabet, and conjectured that a polynomial time $O(\log n)$ -approximation algorithm exists. We refute the conjecture by showing that, for every constant $\varepsilon > 0$, no polynomial time $n^{1-\varepsilon}$ -approximation is possible unless $P = NP$. This together with the simple $O(n)$ -approximation algorithm gives a sharp threshold result for the shortest reset word problem.

The mathematical motivation and its algorithmic version considered in this paper are closely connected, although not in a very formal sense. All known methods for proving bounds on the length of the shortest reset word are actually based on explicitly computing a short reset word (in polynomial time). The best known method constructs a reset word of length $\frac{n^3-n}{6}$, while the (most likely) true upper bound is just $(n-1)^2$, which is smaller by a factor of roughly $\frac{n}{6}$. Similarly, the best known (polynomial time) approximation algorithm achieves $O(n)$ -approximation. Hence it is reasonable to believe that an $o(n)$ -approximation algorithm could be used to significantly improve the upper bound on the length of the shortest synchronizing word to $o(n^3)$. In this context, our result suggests that improving the bound on the length of the shortest synchronizing word to $O(n^{3-\varepsilon})$ requires non-constructive tools.

The main insight is to start with the PCP theorem. We recall the notion of constraint satisfaction problems, and using the result of Håstad and Zuckerman provide a class of hard instances of such problems with specific properties tailored to our particular application. Then, we show how to appropriately translate such a problem into a synchronizing automaton.

Organization of the Paper. We provide the necessary definitions and the background on finite automata in the preliminaries. We also introduce the notion of probabilistically checkable proofs and state the PCP theorem, then define constraint satisfaction problems and their basic parameters.

In the next three sections we gradually move towards the main result. In Sect. 3 we prove that $(2-\varepsilon)$ -approximation of the shortest reset word is NP-hard. In Sect. 4 we strengthen this by showing that, for a small fixed $\varepsilon > 0$, n^ε -approximation is also NP-hard. Finally, in Sect. 5, we provide more background on probabilistically checkable proofs and free bit complexity, and prove that, for every $\varepsilon > 0$, even $n^{1-\varepsilon}$ -approximation is NP-hard. Even though the final result subsumes Sects. 3 and 4, this allows us to gradually introduce the new components.

2 Preliminaries

DFA. A deterministic finite automaton (in short, an automaton) is a triple $A = (Q, \Sigma, \delta)$, where Q is a nonempty finite set of states, Σ is a nonempty finite alphabet, and δ is a transition function $\delta : Q \times \Sigma \rightarrow Q$. In the usual definition one includes additionally a starting state and a set of accepting states, which are irrelevant in our setting. Equivalently, we can treat an automaton as a collection of $|\Sigma|$ transformations of a finite set Q . We consider words over Σ , which are finite sequences of letters (elements of Σ). The empty word is denoted by ε , the set of words of length n by Σ^n , and the set of all words by Σ^* . For $w \in \Sigma^*$, $|w|$ stands for the length of w and w_i is the i -th letter of w , for any $i \in \{1, 2, \dots, |w|\}$.

If $A = (Q, \Sigma, \delta)$ is an automaton, then we naturally extend δ from single letters to whole words by defining $\delta(q, \varepsilon) = q$ and $\delta(q, wa) = \delta(\delta(q, w), a)$. For $P \subseteq Q$ we denote by $\delta(P, w)$ the image of P under $\delta(\cdot, w)$.

Synchronizing Automata. An automaton $A = (Q, \Sigma, \delta)$ is synchronizing if there exists a word w for which $|\delta(Q, w)| = 1$. Such w is then called a synchronizing (or reset) word and the length of a shortest such word is denoted by $\text{Syn}(A)$. One can check if an automaton is synchronizing in polynomial time by verifying that every pair of states can be synchronized to a single state.

SYNAPPX(Σ, α)

Given a synchronizing n -state automaton A over an alphabet Σ , find a word of length at most $\alpha \cdot \text{Syn}(A)$ synchronizing A . Here both α and $|\Sigma|$ can be a function of n .

We are interested in solving **SYNAPPX**(Σ, α) in polynomial time, with α as small as possible.

Alphabet Size. In the general case, the size of the alphabet can be arbitrary. Our construction will use $\Sigma = \{0, 1, 2\}$, which can be then reduced to the binary alphabet using the method of Berlinkov [7], which is based on encoding every letter in binary and adding some intermediate states.

Lemma 1 (Lemma 7 of [7]). *Suppose **SYNAPPX**($\{0, 1\}, n^\alpha$) can be solved in polynomial time for some $\alpha \in (0, 1)$, then so can be **SYNAPPX**($\Sigma, O(n^\alpha)$) for any Σ of constant size.*

PCP Theorems. We briefly introduce the notion of *Probabilistically Checkable Proofs* (PCPs). For a comprehensive treatment refer to [5] or [2].

A polynomial-time probabilistic machine V is called a $(p(n), r(n), q(n))$ -PCP verifier for a language $L \subseteq \{0, 1\}^*$ if:

- for an input x of length n , given random access to a “proof” $\pi \in \{0, 1\}^*$, V uses at most $r(n)$ random bits, accesses at most $q(n)$ locations of π , and outputs 0 or 1 (meaning “reject” or “accept” respectively),
- if $x \in L$ then there is a proof π , such that $\Pr[V(x, \pi) = 1] = 1$,
- if $x \notin L$ then for every proof π , $\Pr[V(x, \pi) = 1] \leq p(n)$.

We consider only *nonadaptive* verifiers, meaning that the subsequently accessed locations depend only on the input and the random bits, and not on the previous answers, hence we can think that V specifies at most $q(n)$ locations and then receives a sequence of bits encoding all the answers. $p(n)$ from the above definition is often called the *soundness* or the *error probability*. In some cases, also the *proof length* is important. For a fixed input x of length n the proof length is the total number of distinct locations queried by V over all possible $2^{r(n)}$ runs of V (on different sequences of $r(n)$ random bits). The proof length is always at most $q(n) \cdot 2^{r(n)}$, and such a bound is typically sufficient for applications, however in some cases we desire PCP-verifiers with smaller proof length.

The set of languages for which there exists a (p, r, q) -PCP verifier is denoted by $\text{PCP}_p[r, q]$.

Theorem 2 (PCP Theorem [3, 4]). $\text{NP} = \text{PCP}_{1/2}[O(\log n), O(1)]$.

Constraint Satisfaction Problems. We consider *Constraint Satisfaction Problems* (CSPs) over boolean variables. An instance of a general CSP over N boolean variables x_1, x_2, \dots, x_N is a collection of M boolean constraints $\phi = (C_1, C_2, \dots, C_M)$, where a boolean constraint is just a function $C : \{0, 1\}^N \rightarrow \{0, 1\}$. A boolean assignment $v : \{0, 1\}^N \rightarrow \{0, 1\}$ satisfies a constraint C if $C(v) = 1$, and ϕ is satisfiable if there exists an assignment $v : \{0, 1\}^N \rightarrow \{0, 1\}$ such that $C_i(v) = 1$ for all $i = 1, 2, \dots, M$. We define $\text{Val}(\phi)$ to be the maximum fraction of constraints in ϕ which can be satisfied by a single assignment. In particular $\text{Val}(\phi) = 1$ iff ϕ is satisfiable.

We consider computational properties of CSPs. We are mainly interested in CSPs, where every N -variable constraint has description of size $\text{poly}(N)$ (as opposed to the naive representation using 2^N bits). A natural class of such CSPs are CNF-formulas, where every constraint is a clause being a disjunction of N literals, thus described in $O(N)$ space. Another important class are qCSPs, where every constraint *depends* only on at most q variables. Such a constraint can be described using $\text{poly}(N, 2^q)$ space, which is polynomial whenever $q = O(\log N)$. Formally, we say that a clause C depends on variable x_i if there exists an assignment $v \in \{0, 1\}^N$ such that $C(v)$ changes after modifying the value of x_i and keeping the remaining variables intact. We define V_C to be the set of all such variables. It is easy to see that $\phi(C)$ is determined as soon as we assign the values to all variables in V_C . Finally, the following class will be of interest to us.

Definition 3. Let C be an N -variable constraint and let V_C be the set of variables on which C depends. Consider all $2^{|V_C|}$ assignments $\{0, 1\}^{V_C} \rightarrow \{0, 1\}$. If only K of such assignments satisfy C , we write $\text{Fsat}(C) \leq K$. $\text{Fsat}(\phi) \leq K$ if $\text{Fsat}(C) \leq K$ for every constraint C in ϕ .

According to the above definition, if ϕ is a qCSP instance then $\text{Fsat}(\phi) \leq 2^q$. A constraint C such that $\text{Fsat}(C) \leq K$ can be described by its set V_C and a list of at most K assignments to the variables in V_C satisfying C . Thus the description is polynomial in N and K . We will consider CSPs ϕ with $\text{Fsat}(\phi) \leq \text{poly}(N)$ and always assume that they are represented as just described.

3 Simple Hardness Result

We start with a simple introductory result, which is that for any fixed constant $\varepsilon > 0$, it is NP-hard to find for a given n -state synchronizing automaton A a synchronizing word w such that $|w| \leq (2 - \varepsilon) \cdot \text{Syn}(A)$. The final goal is to prove a much stronger result, but the basic construction presented in this section is the core idea further developed in the subsequent sections. The construction is not the simplest possible, nor the most efficient in the number of states of the resulting automaton, but it provides good intuitions for the further proofs. For a simpler construction in this spirit see [6].

Theorem 4. *For every constant $\varepsilon > 0$, $\text{SYNAPPX}(\{0, 1, 2\}, 2 - \varepsilon)$ is not solvable in polynomial time, unless $\text{P} = \text{NP}$.*

Idea. Fix $\varepsilon > 0$. We will reduce 3-SAT to our problem, that is, show that an algorithm solving $\text{SYNAPPX}(\{0, 1, 2\}, 2 - \varepsilon)$ can be used to decide satisfiability of 3-CNF formulas. This will stem from the following reduction. For a given N -variable 3-CNF formula ϕ consisting of M clauses we can build in polynomial time a synchronizing automaton A_ϕ such that:

1. if ϕ is satisfiable then $\text{Syn}(A_\phi) \approx N$,
2. if ϕ is not satisfiable then $\text{Syn}(A_\phi) \geq 2N$.

This implies Theorem 4, since applying an $(2 - \varepsilon)$ -approximation algorithm to A_ϕ allows us to find out whether ϕ is satisfiable or not.

Construction. Let $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_M$ be a 3-CNF formula with N variables x_1, x_2, \dots, x_N and M clauses. We want to build an automaton $A_\phi = (\{0, 1, 2\}, Q, \delta)$ with properties as described above. A_ϕ consists of M gadgets, one for each clause in ϕ , and a single sink state s . All letters leave s intact, that is, $\delta(s, 0) = \delta(s, 1) = \delta(s, 2) = s$. We describe now a gadget for a fixed clause C .

The gadget built for a clause C can be essentially seen as a tree with 8 leaves. Each leaf corresponds to one of the assignments to 3 variables appearing in C . First we introduce the uncompressed version of the gadget. Take all possible 2^N assignments and form a full binary tree of height N . Every edge in the tree is directed from a parent to its child and has a label from $\{0, 1\}$. Every assignment naturally corresponds to a leaf in the tree. We could potentially use such a tree as the gadget, except that its size is exponential. We will fix this by merging isomorphic subtrees to obtain a tree of size linear in N .

Let us denote by L_0, L_1, \dots, L_N the vertices at levels $0, 1, \dots, N$, respectively, so that $L_0 = \{r\}$, where r is the root, and L_N is the set of leaves.

Suppose that the variable x_k does not occur in C . Take any vertex $v \in L_{k-1}$ and denote the subtrees rooted at its children by T_0 and T_1 . It is easy to see that T_0 and T_1 are isomorphic and can be merged, so that we have two edges outgoing from v , labeled by 0 and 1, respectively, and both leading to the same vertex v' , which is the root of T_0 . We continue the merging until there are no more

such vertices, which can be seen as “compressing” the tree. The last layer L_N of the resulting compressed tree contains states of the form q_N^w , where $w \in \{0, 1\}^3$ is some boolean assignment, and $\delta(q_N^w, 0) = \delta(q_N^w, 1) = s$ if w satisfies C and $\delta(q_N^w, 0) = \delta(q_N^w, 1) = r$ otherwise.

The above defined *tree-gadget* will be further denoted by T_C , and its root q_0^ε will be usually referred to as r . To complete the definition, we set $\delta(q, 2) = r$ for every $q \in T_C$. See Fig. 1 for an example.

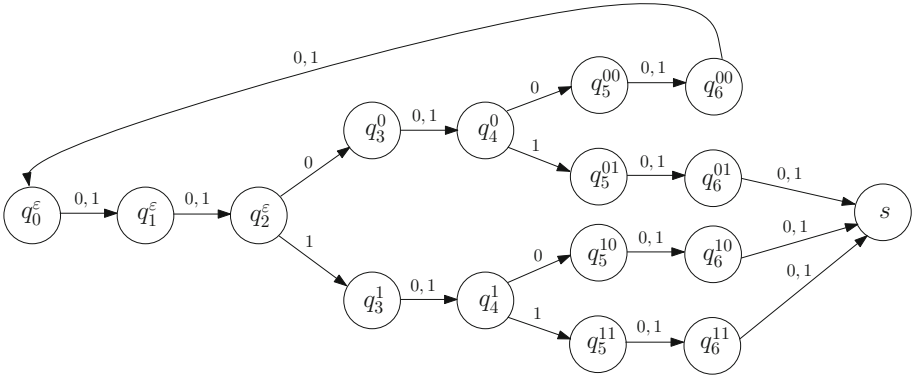


Fig. 1. Tree-gadget T_C constructed for a clause $C = x_3 \vee x_5$ and $N = 6$.

The automaton A_ϕ consists of M disjoint tree-gadgets $T_{C_1}, T_{C_2}, \dots, T_{C_M}$ and a single “sink state” s . Formally, its set of states is $Q = \sum_{i=1}^M T_{C_i} \cup \{s\}$ and the transition δ is defined above for every tree-gadget and the sink state s .

Properties of A_ϕ . The following properties of A_ϕ can be established.

Proposition 5. Consider a tree-gadget T_C with root r constructed for a clause C . If C depends only on variables $x_{j_1}, x_{j_2}, x_{j_3}$ then for any binary assignment $v \in \{0, 1\}^N$ we have $\delta(r, v) = q_N^w$, where $w = v_{j_1}v_{j_2}v_{j_3}$.

Proposition 5 immediately yields the following.

Corollary 6. Consider a tree-gadget T_C constructed for a clause C , and let $w = vc$ be a binary word with $|v| = N$ and $c \in \{0, 1\}$. If v is an assignment satisfying C then $\delta(r, w) = s$, otherwise $\delta(r, w) = r$.

Since s is a sink state, synchronizing A_ϕ is equivalent to pushing all of its states into s . Actually, it is enough to consider how to synchronize the set $R = \{r_1, r_2, \dots, r_M\}$, where r_i is the root of the i -th tree-gadget T_{C_i} . This is because $\delta(T_{C_i}, 2) = \{r_i\}$ for every i , hence one application of letter 2 “synchronizes” every gadget to its root and then it is enough to synchronize the roots.

It is already easy to see that A_ϕ is always synchronizing, because we can synchronize gadgets one by one. The following lemma says that in case when ϕ is satisfiable, we can synchronize A_ϕ very quickly.

Lemma 7. *If ϕ is satisfiable and $v \in \{0, 1\}^N$ is a satisfying assignment, then the word $w = 2v0$ synchronizes A_ϕ . Therefore $\text{Syn}(A_\phi) \leq N + 2$.*

Our next goal is to show that whenever ϕ is not satisfiable, A_ϕ cannot be synchronized quickly. To this end, we prove the following.

Lemma 8. *Suppose there exists a binary word of length less than $2N + 2$ synchronizing R to $\{s\}$. Then ϕ is satisfiable.*

By Lemma 8, if there is no short binary word synchronizing R then ϕ is not satisfiable. Using letter 2 does not help at all in synchronizing R as shown below.

Lemma 9. *Suppose there exists a word $w \in \{0, 1, 2\}^*$ synchronizing R to $\{s\}$. Then there is a word $w' \in \{0, 1\}^*$ of length at most $|w|$ synchronizing R to $\{s\}$.*

We are now ready to prove Theorem 4.

Proof (of Theorem 4). Fix $\varepsilon > 0$ and suppose we can solve $\text{SYNAPPX}(\{0, 1, 2\}, 2 - \varepsilon)$ in polynomial time. We will show that we can solve 3-SAT in polynomial time. Let ϕ be any 3-CNF formula. We construct A_ϕ and approximate its shortest reset word within a factor of $2 - \varepsilon$. By Lemma 7 if ϕ is satisfiable then $\text{Syn}(A_\phi) \leq N + 2$, and if ϕ is not satisfiable then by Lemmas 8 and 9 we have $\text{Syn}(A_\phi) \geq 2N + 2$. Hence, $(2 - \varepsilon)$ -approximation allows us to distinguish between those two cases in polynomial time. \square

4 Hardness with Ratio n^ε

In this section we show that it is possible to achieve a stronger hardness result using essentially the same reduction, but from a different problem. The problem we reduce from is CSP with some specific parameters. Its hardness is proved by suitably amplifying the error probability in the classical PCP theorem. We provide the details below.

PCP, qCSP and Probability Amplification. We want to obtain a hard boolean satisfaction problem, which allows us to perform more efficient reductions to SYNAPPX . The usual source of such problems are PCP theorems, the most basic one asserting that $\text{NP} = \text{PCP}_{1/2}[O(\log n), O(1)]$. By sequential repetition we can obtain verifiers erring with much lower probability, i.e., $\text{NP} = \text{PCP}_\varepsilon[O(\log n), O(1)]$ for any fixed $\varepsilon \in (0, 1)$. Combining such a verifier with the construction of A_ϕ described in the previous section yields that it is NP-hard to approximate the shortest reset word within a factor of α , for any constant α . However, we aim for a stronger n^ε -hardness for some $\varepsilon > 0$. To this end we need to construct PCP verifiers with subconstant error.

Sequential repetition used to reduce the error probability as explained above has severe limitations. We want the error probability to be $\approx n^{-1}$. This requires $\Theta(\log n)$ repetitions, each consuming fresh $O(\log n)$ random bits, and results in a verifier with the error probability bounded by n^{-1} using $O(\log n)$ queries.

The total number of used random bits is then $r = \Theta(\log^2 n)$, which is too much, since the size of the automaton polynomially depends on 2^r . Fortunately, the amount of used random bits can be reduced using the standard idea of a random walk on an expander, resulting in the following theorem.

Theorem 10 (Subconstant Error PCP). $\text{NP} \subseteq \text{PCP}_{1/n}[O(\log n), O(\log n)]$.

We use Theorem 10 to prove the following.

Theorem 11. *There exists a polynomial time reduction f , which takes a 3-CNF n -variable formula ϕ and returns a qCSP instance $f(\phi)$ with $q = O(\log n)$, such that:*

- if ϕ is satisfiable then $\text{Val}(f(\phi)) = 1$,
- if ϕ is not satisfiable then $\text{Val}(f(\phi)) \leq \frac{1}{n}$.

Construction. Let ϕ be an N -variables qCSP instance with M clauses and $q = O(\log N)$. We want to construct a synchronizing automaton A_ϕ , such that the length of its shortest reset word allows us to reconstruct $\text{Val}(\phi)$ up to some error.

The construction of A_ϕ is exactly the same as the one given for 3-CNF instances in Sect. 3. For a q -constraint C we build a tree-gadget T_C with 2^q leaves, each corresponding to an assignment to the variables C depends on. As previously, the automaton has one sink state s and M tree-gadgets, one for every constraint. The construction still takes just polynomial time, the size of the automaton is polynomial in M, N and $2^q = \text{poly}(N)$.

Properties of A_ϕ . Similarly as in the previous sections, the following properties of A_ϕ can be established.

Lemma 12. *Let ϕ be a N -variable qCSP instance. If ϕ is satisfiable and $v \in \{0, 1\}^N$ is a satisfying assignment, then the word $w = 2v0$ synchronizes A_ϕ . Therefore $\text{Syn}(A_\phi) \leq N + 2$.*

For the case when ϕ is not satisfiable we need a stronger statement than the one from Lemma 8.

Lemma 13. *Let ϕ be a N -variable qCSP instance. If w synchronizes A_ϕ then $|w| \geq \frac{1}{\text{Val}(\phi)}(N + 1)$.*

Now we are ready to prove the main theorem of this section.

Theorem 14. *There exists a constant $\varepsilon > 0$, such that $\text{SYNAPPX}(\{0, 1, 2\}, n^\varepsilon)$ is not solvable in polynomial time, unless $\text{P} = \text{NP}$.*

Proof. We reduce 3-SAT to $\text{SYNAPPX}(\{0, 1, 2\}, n^\varepsilon)$, for some constant $\varepsilon > 0$. Let ϕ be an n -variable 3-CNF formula ϕ . We use Theorem 11 to obtain a qCSP instance $f(\phi)$ on N variables and then convert it into a G -state automaton $A_{f(\phi)}$. If ϕ is satisfiable, then by Lemma 12 $\text{Syn}(A_{f(\phi)}) \leq N + 2$. On the other hand, if ϕ is not satisfiable, then $\text{Val}(f(\phi)) \leq 1/n$, hence by Lemma 13 $\text{Syn}(A_{f(\phi)}) \geq n(N + 1)$. The ratio between those two quantities is $\frac{n(N+1)}{N+2} = \Omega(n)$.

It remains to show that $n = \Omega(G^\varepsilon)$ for some constant $\varepsilon > 0$. In other words, we need to show that G is polynomial in n . This holds, because f is a polynomial time reduction, hence $N, M = \text{poly}(n)$ and the size of $A_{f(\phi)}$ is polynomial with respect to $N, M, 2^q$, but $q = O(\log N) = O(\log n)$ so $2^q = \text{poly}(n)$. \square

Remark 15. By keeping track of all the constants, one can obtain n^ε -hardness for $\varepsilon \approx 0.0095$, but this is anyway subsumed by the next section.

5 Hardness with Ratio $n^{1-\varepsilon}$

In this section we prove the main result of the paper. It is not enough to use the reasoning from the previous section and simply optimize the constants. In fact, the strongest hardness result that we can possibly obtain by applying Theorem 10 is n^ε for some tiny constant $\varepsilon > 0$. This stems from the fact that in our reduction we require the number of queries q to be logarithmic in the size of the instance. If this is not the case, then the reduction takes superpolynomial time. However, the crucial observation is that the reduction can be modified so that we do not need the query complexity of the verifier to be logarithmic. It suffices that the free bit complexity (defined below) is logarithmic.

Theorem 16. *For every constant $\varepsilon > 0$, $\text{SYNAPPX}(\{0, 1, 2\}, n^{1-\varepsilon})$ is not solvable in polynomial time, unless $\text{P} = \text{NP}$.*

Free Bit Complexity and Stronger PCP Theorems. Let us first briefly introduce the notion of free bit complexity. For a comprehensive discussion see [5].

Definition 17. *Consider a PCP verifier V using r random bits on any input x . For a fixed input x and a sequence of random bits $R \in \{0, 1\}^r$, define $G(x, R)$ to be the set of sequences of answers to the questions asked by V , which result in an acceptance. We say that V has free bit complexity f if $|G(x, R)| \leq 2^f$ and there is a polynomial time algorithm which computes $G(x, R)$ for given x and R .*

The set of languages for which there exists a verifier with soundness p , free bit complexity f , and proof length ℓ is denoted by $\text{FPCP}_p[r, f, \ell]$.

Håstad in his seminal work [12] proved that approximating the maximum clique within the factor $n^{1-\varepsilon}$ is hard, for every $\varepsilon > 0$. To obtain this result he constructs PCP verifiers with arbitrarily small *amortized free bit complexity*¹. We state his result in a more recent and stronger version [13]:

¹ Amortized free bit complexity is a parameter of a PCP verifier which essentially corresponds to the ratio between the free bit complexity and the logarithm of error probability.

Theorem 18. *For every $\varepsilon > 0$, there exist constants $t \in \mathbb{N}$ and $\alpha, \beta > 0$ such that $\text{NP} \subseteq \text{FPCP}_{2^{-t}}[\beta \log n, \varepsilon \cdot t, n^\alpha]$.*

In the next step we need to amplify the error probability, as we did in the previous section. It turns out that the amplification using expander walks is too weak for our purpose. Håstad [12], following the approach of Bellare et al. [5], uses sequential repetition together with a technique to reduce the demand for random bits. (See Proposition 11.2, Corollary 11.3 in [5].) Unfortunately, this procedure involves randomization, so his MaxClique hardness result holds under the assumption that $\text{ZPP} \neq \text{NP}$.

In his breakthrough paper Zuckerman [20] showed how to derandomize Håstad’s MaxClique hardness result by giving a deterministic method for amplifying the error probability of PCP verifiers. He constructs very efficient randomness extractors, which then by known reductions allow to perform error amplification. One can conclude the following result from Theorem 18 and his result (see also Lemma 6.4 and Theorem 1.1 in [20]).

Theorem 19. *For every $\varepsilon > 0$, there exist $c, \alpha > 0$ such that for $t = c \log n$ it holds that $\text{NP} \subseteq \text{FPCP}_{2^{-t}}[(1 + \varepsilon)t, \varepsilon t, n^\alpha]$.*

Based on the above theorem, we can prove the following very strong analogue of Theorem 11.

Theorem 20. *For every $\varepsilon > 0$, there exists a polynomial time reduction f , which takes an n -variable 3-CNF formula ϕ and returns an N -variable CSP instance $f(\phi)$ with M constraints, such that:*

- $N \leq M^\varepsilon$,
- if ϕ is satisfiable then $\text{Val}(f(\phi)) = 1$,
- if ϕ is not satisfiable then $\text{Val}(f(\phi)) \leq \frac{1}{M^{1-\varepsilon}}$,
- $\text{Fsat}(f(\phi)) \leq M^\varepsilon$.

Construction. Let ϕ be an N -variable CSP instance with M constraints such that $\text{Fsat}(\phi) \leq K$ (for a parameter K to be chosen later). We want to construct an automaton \hat{A}_ϕ of size polynomial in K and the size of ϕ such that $\text{Syn}(\hat{A}_\phi) \approx \frac{N}{\text{Val}(\phi)}$.

Using A_ϕ as in the previous sections gives an automaton of superpolynomial size, so we need to tweak it. Take any constraint C and suppose it depends on q variables. Consider the tree-gadget T_C built for C . We cannot assume that $q = O(\log n)$ as in the Sect. 4. In consequence, T_C can be of exponential size, because the only possible bound on its number of leaves is 2^q . However, we have a bound K on the number of essentially satisfying assignments. We will modify the definition of T_C , so that its size depends polynomially on K rather than 2^q .

Observe that in the original construction, at most K out of 2^q leaves of T_C correspond to satisfying assignments (think of K much smaller than 2^q). Imagine for a moment a subtree of T_C corresponding to the at most K satisfying assignments. The size of such subtree is at most NK , but it is not yet a good

candidate for our gadget, because some transitions are not well defined. However, this is not difficult to fix to obtain an equivalent *compressed tree-gadget* \hat{T}_C as follows. Consider the example from Fig. 2 and let w be a state reached from the leftmost state after applying 0. All leaves of the subtree rooted at w correspond to non-satisfying assignments, hence the whole subtree can be replaced by a path of appropriate length. We repeat such replacement for every node such that its whole subtree contains only leaves corresponding to non-satisfying assignments, but the subtree of its parent does not.

The automaton \hat{A}_ϕ is built analogously to A_ϕ , with the crucial difference that we use \hat{T}_C instead of T_C . One can see that \hat{T}_C can be constructed in time polynomial in its size by proceeding from the root down to the leaves. Furthermore, the resulting automaton \hat{A}_ϕ is small.

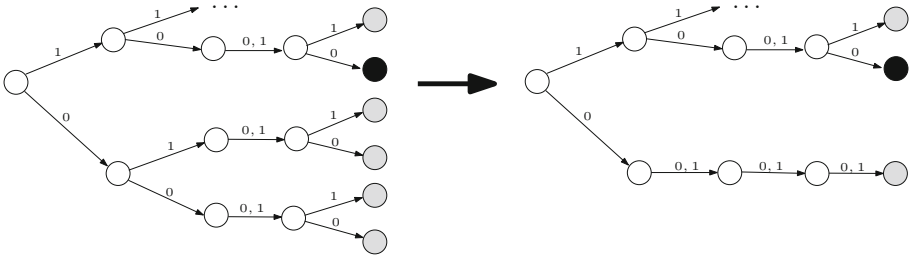


Fig. 2. Compressing the tree. The grey and black leaves correspond to non-satisfying and satisfying assignments, respectively (Color figure online).

Lemma 21. *Suppose ϕ is an N -variable CSP instance with M constraints and $\text{Fsat}(\phi) \leq K$. Then the size of \hat{A}_ϕ is $O(MN^2K)$.*

Properties of \hat{A}_ϕ . The lemma below summarizes the properties of \hat{A}_ϕ . Its proof is very similar to the proofs of the lemmas summarizing the properties of A_ϕ and hence skipped.

Lemma 22. *Let ϕ be an N -variable CSP instance with M constraints and $\text{Fsat}(\phi) \leq K$. Then \hat{A}_ϕ is a synchronizing automaton of size $O(MN^2K)$, which can be constructed in polynomial time. Furthermore, if ϕ is satisfiable then $\text{Syn}(\hat{A}_\phi) \leq N + 2$ and otherwise $\text{Syn}(\hat{A}_\phi) \geq \frac{N+1}{\sqrt{\text{Val}(\phi)}}$.*

Proof (of Theorem 16). Fix any $\varepsilon > 0$. We reduce 3-SAT to $\text{SYNAPPX}(\{0, 1, 2\}, n^\varepsilon)$. Let ϕ be an n -variable 3-CNF formula. Then by Theorem 20 we can construct an N -variable CSP instance $f(\phi)$ with $M = \text{poly}(n)$ constraints and $\text{Fsat}(f(\phi)) \leq K = M^\varepsilon$, where $N \leq M^\varepsilon$. We know that if ϕ is satisfiable then $f(\phi)$ is satisfiable as well and if ϕ is not satisfiable then $\text{Val}(f(\phi)) \leq \frac{1}{M^{1-\varepsilon}}$. Then by Theorem 22 we can construct $\hat{A}_{f(\phi)}$, which is an automaton of size

$O(MKN^2) = O(M^{1+3\epsilon})$. If ϕ is satisfiable, $\text{Syn}(\hat{A}_{f(\phi)}) \leq N + 2$ and if ϕ is not satisfiable then $\text{Syn}(\hat{A}_{f(\phi)}) \geq \frac{N+1}{\text{Val}(f(\phi))}$. The ratio of those two bounds is:

$$\frac{N+1}{(N+2)\text{Val}(f(\phi))} = \Theta\left(\frac{1}{\text{Val}(f(\phi))}\right) = \Theta(M^{1-\epsilon})$$

The size of the automaton $\hat{A}_{f(\phi)}$ is $G = O(M^{1+3\epsilon})$, so the above ratio can be related to the size of the automaton as $\Omega\left(G^{\frac{1-\epsilon}{1+3\epsilon}}\right) = \Omega(G^{1-4\epsilon})$. Hence assuming $P \neq NP$, approximating the shortest reset word within ratio $G^{1-4\epsilon}$ in polynomial time is not possible. \square

References

1. Ananichev, D.S., Volkov, M.V.: Synchronizing generalized monotonic automata. *Theor. Comput. Sci.* **330**(1), 3–13 (2005)
2. Arora, S., Barak, B.: *Computational Complexity: A Modern Approach*, 1st edn. Cambridge University Press, Cambridge (2009)
3. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. *J. ACM* **45**(3), 501–555 (1998)
4. Arora, S., Safra, S.: Probabilistic checking of proofs: a new characterization of NP. *J. ACM* **45**(1), 70–122 (1998)
5. Bellare, M., Goldreich, O., Sudan, M.: Free bits, PCPs, and nonapproximability–towards tight results. *SIAM J. Comput.* **27**, 804–915 (1998)
6. Berlinkov, M.V.: Approximating the minimum length of synchronizing words is hard. *Theor. Comp. Sys.* **54**(2), 211–223 (2014)
7. Berlinkov, M.V.: On two algorithmic problems about synchronizing automata. In: Shur, A.M., Volkov, M.V. (eds.) *DLT 2014*. LNCS, vol. 8633, pp. 61–67. Springer, Heidelberg (2014)
8. Eppstein, D.: Reset sequences for monotonic automata. *SIAM J. Comput.* **19**, 500–510 (1990)
9. Feige, U.: A threshold of $\ln n$ for approximating set cover. *J. ACM* **45**(4), 634–652 (1998)
10. Gerbush, M., Heeringa, B.: Approximating minimum reset sequences. In: Domaratzki, M., Salomaa, K. (eds.) *CIAA 2010*. LNCS, vol. 6482, pp. 154–162. Springer, Heidelberg (2011)
11. Grech, M., Kisielewicz, A.: The Černý conjecture for automata respecting intervals of a directed graph. *Discrete Mathematics & Theoretical Computer Science* **15**(3), 61–72 (2013)
12. Hastad, J.: Clique is hard to approximate within $n^{1-\epsilon}$. In: *Proceedings of the 37th Annual Symposium on Foundations of Computer Science, FOCS 1996*, pp. 627–636 (1996)
13. Hastad, J., Khot, S.: Query efficient PCPs with perfect completeness. In: *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pp. 610–619, October 2001
14. Kari, J.: Synchronizing finite automata on Eulerian digraphs. *Theor. Comput. Sci.* **295**, 223–232 (2003)

15. Olschewski, J., Ummels, M.: The complexity of finding reset words in finite automata. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 568–579. Springer, Heidelberg (2010)
16. Pin, J.: On two combinatorial problems arising from automata theory. In: Combinatorial Mathematics Proceedings of the International Colloquium on Graph Theory and Combinatorics, vol. 75, pp. 535–548. North-Holland (1983)
17. Rystsov, I.: Reset words for commutative and solvable automata. *Theor. Comput. Sci.* **172**(1–2), 273–279 (1997)
18. Steinberg, B.: The Černý conjecture for one-cluster automata with prime length cycle. *Theor. Comput. Sci.* **412**(39), 5487–5491 (2011)
19. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)
20. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. In: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, STOC 2006, pp. 681–690 (2006)

Finitary Semantics of Linear Logic and Higher-Order Model-Checking

Charles Grellois^(✉) and Paul-André Mellies^(✉)

Laboratoire PPS, Université Paris Diderot, Sorbonne Paris Cité, France
{grellois,mellies}@pps.univ-paris-diderot.fr

Abstract. In this paper, we explain how the connection between higher-order model-checking and linear logic recently exhibited by the authors leads to a new and conceptually enlightening proof of the selection problem originally established by Carayol and Serre using collapsible push-down automata. The main idea is to start from an infinitary and colored relational semantics of the λY -calculus formulated in a companion paper, and to replace it by a finitary counterpart based on finite prime-algebraic lattices. Given a higher-order recursion scheme \mathcal{G} , the finiteness of its interpretation in the resulting model enables us to associate to any MSO formula φ a higher-order recursion scheme \mathcal{G}_φ resolving the selection problem.

Keywords: Higher-order model-checking · Linear logic · Selection problem · Finitary semantics · Parity games

1 Introduction

Higher-order recursion schemes (HORS) provide an abstract model of computation which appears to be perfectly adapted for the task of model-checking functional programs. Indeed, Knapik, Niwinski and Urzyczyn established in [7] that for $n \geq 1$, the trees generated by order- n safe recursion schemes are exactly those that are generated by order- n pushdown automata, and further, that they have decidable MSO theories. The MSO-decidability result for safe HORS was then extended a few years later to all HORS by Ong [9]. However, the MSO-decidability theorem established by the four authors focuses on the decidability of a “local” model-checking problem:

Suppose given a HORS \mathcal{G} which generates an infinite tree $\langle \mathcal{G} \rangle$. Is it possible to decide for every MSO-formula φ whether the formula is valid at the root of the infinite tree $\langle \mathcal{G} \rangle$.

The MSO-decidability result means that the answer to this question is positive. A more difficult “global” model-checking problem called the *selection problem* in literature is to understand whether:

Given a HORS \mathcal{G} and a MSO-formula $\exists X \varphi[X]$ holding at the root of the infinite tree $\langle \mathcal{G} \rangle$, is it possible to compute a HORS \mathcal{G}_φ generating a marked version $\langle \mathcal{G}_\varphi \rangle$ of the original tree $\langle \mathcal{G} \rangle$, and such that the set of its marked nodes is a witness U satisfying the MSO-formula $\varphi[X]$.

Quite strikingly, Carayol and Serre established in a recent paper [2] that the answer to this question is positive. They also noticed that the selection problem follows from a purely automata-theoretic property of HORS, which was established by Haddad in his PhD thesis [6]:

Given a HORS \mathcal{G} and an alternating parity tree automaton \mathcal{A} with the same ranked alphabet, for every state q of the automaton \mathcal{A} accepted by the tree $\langle \mathcal{G} \rangle$, it is possible to compute a HORS \mathcal{G}_q generating an accepting run-tree $\langle \mathcal{G}_q \rangle$ of the automaton \mathcal{A} on the tree $\langle \mathcal{G} \rangle$ with initial state q .

Of course, the run-tree $\langle \mathcal{G}_q \rangle$ generated by the HORS \mathcal{G}_q provides a witness of the fact that the state q is accepting. But not only that: thanks to the equivalence between MSO-formulas and alternating parity tree automata (APT), the fact that the HORS \mathcal{G}_q selects a *specific* run-tree $\langle \mathcal{G}_q \rangle$ among all the run-trees with initial state q provides a solution to the “selection problem”. The idea is simply to extract from the run-tree $\langle \mathcal{G}_q \rangle$ a specific witness X for the MSO-formula $\exists X \varphi[X]$ satisfied by the tree $\langle \mathcal{G} \rangle$.

In this article, we will show how to establish the existence of such a “higher-order recursive” run-tree $\langle \mathcal{G}_q \rangle$ from purely denotational arguments, based on a new and fundamental connection with linear logic developed by the authors in a series of recent papers [4,5]. In these papers, an infinitary and colored variant of the traditional semantics of linear logic is constructed, see [4] for details, and shown to compute in a compositional way the set of accepting states of an alternating parity tree automaton, see [5] for details. Despite the conceptual clarification this approach provides to higher-order model-checking, this semantic account does not lead to any decidability result. The reason is that the relational semantics of linear logic is a *quantitative* semantics, where finite types are interpreted as infinitary objects. In order to establish decidability results, one thus needs to shift to *qualitative* semantics where the interpretation of finite types remains finite. This is precisely the purpose of the present paper: by shifting from the relational semantics developed in [4,5] to the qualitative semantics of linear logic provided by prime-algebraic lattices, we are able to establish advanced decidability results like the theorem just mentioned by Carayol, Haddad and Serre. This is the first time, to our knowledge, that such a strong and natural connection between model-checking and the most contemporary tools of semantics (linear logic, relational semantics) is exhibited.

Plan of the Paper. We start by recalling in Sect. 2 the notion of higher-order recursion scheme and its correspondence with the λY -calculus. We then recall in Sect. 3 the notion of alternating parity tree automaton. In Sect. 4, we introduce a *finitary* colored semantics of the λY -calculus, which we use in Sect. 5 to interpret

λ -terms. We define a parameterized fixpoint in this model in Sect. 6, obtaining colored semantics of the λY -calculus. In Sect. 7, we use the finiteness of the model to prove the decidability of the local model-checking and of the selection problem. We finally conclude in Sect. 8.

2 Higher-Order Recursion Schemes and the λY -Calculus

Higher-order Recursion Schemes. The set of simple types of the λ -calculus is generated by the grammar $\sigma, \tau ::= o \mid \sigma \rightarrow \tau$. We write $t :: \sigma$ when a (possibly open) λ -term t has simple type σ . Given a ranked alphabet Σ , a finite set of variables \mathcal{V} , a finite set of simply-typed *non-terminals* \mathcal{N} , and a distinguished non-terminal $S \in \mathcal{N}$, a higher-order recursion scheme (HORS) is the data, for every non-terminal $F \in \mathcal{N}$, of a closed simply-typed λ -term

$$\mathcal{R}(F) = \lambda x_1. \dots \lambda x_n. t \tag{1}$$

of same type as the non-terminal $F \in \mathcal{N}$, with constants in Σ , where $x_i \in \mathcal{V}$ and $t :: o$ is a λ -term of ground type without λ -abstractions. Note that an element $a \in \Sigma$ of arity n is represented as a constant of type $o \rightarrow \dots \rightarrow o \rightarrow o$ with same arity n . For each non-terminal $F \in \mathcal{N}$, the data provided by $\mathcal{R}(F)$ is equivalently represented as a rewrite rule

$$F t_1 \dots t_n \rightarrow_{\mathcal{G}} t[x_i \leftarrow t_i].$$

Every higher-order recursion scheme \mathcal{G} generates a potentially infinite Σ -labelled ranked tree noted $\langle \mathcal{G} \rangle$ and called its *value tree*. This tree is simply obtained by applying an infinite number of times and in a fair way the rewrite rules $\rightarrow_{\mathcal{G}}$ of the HORS \mathcal{G} starting from the start symbol $S \in \mathcal{N}$.

Example 1. Given $\Sigma = \{ \text{if} : 2, \text{data} : 1, \text{Nil} : 0 \}$, consider the HORS \mathcal{G}

$$\begin{cases} \mathbf{S} &= \mathbf{L} \text{ Nil} \\ \mathbf{L} &= \lambda x. \text{if } x \text{ (} \mathbf{L} \text{ (} \mathbf{data} \ x \text{))} \end{cases} \tag{2}$$

which abstracts a simple program whose function **Main** (abbreviated as **S**) calls a function **Listen** (denoted **L**), starting from an empty list. The function **Listen** either returns a stack of data, or receives a new element and pushes it on the current stack. The value tree $\langle \mathcal{G} \rangle$ of this scheme, depicted in Fig. 1, provides an abstraction of the set of potential executions of the program. Note that even though the program **Main** is very simple, its value tree $\langle \mathcal{G} \rangle$ is not regular, since it admits an infinite number of different subtrees.

λ -calculus with Recursion. It is well-known among the specialists of the λ -calculus that higher-order recursion schemes can be nicely represented as simply-typed λ -terms in a λ -calculus extended with a fixpoint operator Y . The resulting λY -calculus is thus defined by adding to the simply-typed λ -calculus, a fixpoint operator Y_σ of type $\sigma \rightarrow \sigma$ together with a rewriting rule for every simple type σ :

$$Y_\sigma M \rightarrow_\delta M (Y_\sigma M)$$

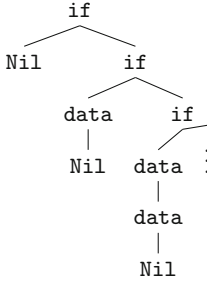


Fig. 1. An order-1 value tree.

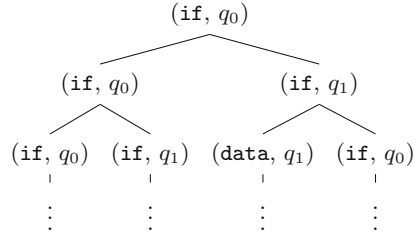


Fig. 2. An APT run-tree.

Proposition 1. *For every HORS \mathcal{G} of ranked alphabet Σ , there exists a closed λY -term $t :: o$ with constants in Σ , such that the λY -term t converges to the value-tree $\langle \mathcal{G} \rangle$ in the traditional sense of Böhm trees in the λY -calculus. Conversely, there exists for every closed λY -term $t :: o$ with constants in Σ a HORS \mathcal{G} of the same ranked alphabet Σ , such that the λY -term t converges to $\langle \mathcal{G} \rangle$.*

An important benefit of this equivalence property is that the λY -calculus is very well understood from the semantic point of view, and thus somewhat simpler to study mathematically speaking than higher-order recursion schemes.

3 MSO and Alternating Parity Tree Automata

As explained in the introduction, there is a beautiful correspondence between the formulas of monadic second-order logic (MSO) and alternating parity tree automata, which we briefly recall here for the sake of completeness.

Proposition 2. *For every ranked alphabet Σ , one has the following equivalence:*

- Every MSO formula φ over Σ -labelled trees can be translated to an APT \mathcal{A}_φ of the same ranked alphabet Σ , such that φ holds at the root of a Σ -labelled tree T iff \mathcal{A}_φ has an accepting run-tree over T from its initial state q_0 .
- Conversely, every APT \mathcal{A} of ranked alphabet Σ can be translated to a MSO formula $\varphi_{\mathcal{A}}$ of the same ranked alphabet, such that for every Σ -labelled tree T , \mathcal{A} has an accepting run-tree over T from its initial state q_0 if and only if the MSO-formula $\varphi_{\mathcal{A}}$ holds at the root of T .

Recall that alternating parity tree automata (APT) are non-deterministic top-down tree automata with the additional ability to *duplicate* or to *erase* subtrees. Typical transitions are thus of the form

$$\delta(q_0, \mathbf{if}) = (2, q_0) \wedge (2, q_1) \qquad \delta(q_1, \mathbf{if}) = (1, q_1) \wedge (2, q_0) \qquad (3)$$

When a node labelled with **if** is visited in state q_0 , its left subtree is “dropped” or “erased” while the right one is “explored twice” or “duplicated”, with q_0 as initial

state in one copy, and q_1 as initial state in the other copy. The second transition does not use alternation, and would be usually written as $(q_1, \mathbf{if}, q_1, q_0) \in \Delta$ in a nondeterministic tree automaton. Run-trees of an alternating parity tree automaton are unranked, and their shape may differ a lot from the original tree. The effect of the transitions (3) over the tree of Fig. 1 is depicted in Fig. 2. In general, a transition is of the shape

$$\delta(q, a) = \bigvee_{i \in I} \bigwedge_{j \in J_i} (d_{i,j}, q_{i,j}) = \bigvee_{i \in I} \varphi_i \tag{4}$$

where the union stands for non-determinism, and the conjunction for alternation: after i is chosen, for every $j \in J_i$, the automaton runs with state $q_{i,j}$ over a copy of its subtree in direction $d_{i,j}$. For every i , we say that φ_i is a *conjunctive clause* of the formula $\delta(q, a)$.

Seen from an automata-theoretic point of view, monadic second-order (MSO) logic is equivalent to the modal μ -calculus. As such it enables one to express safety properties (typically, that a given state “error” is never encountered) as well as liveness properties (typically, that a given state “happy” is visited infinitely often). The safety properties are inductive: it is enough to check that no finite approximation of a computation enters an error state, while the liveness properties are coinductive, since they specify infinitary behaviors. Moreover, MSO logic and the modal μ -calculus are sufficiently expressive to alternate these inductive and coinductive specifications. This alternation is handled by extending APT with a *parity condition* over their run-trees. Alternating parity automata are thus equipped with a *coloring function* $\Omega : Q \rightarrow \mathbb{N}$, which associates a color to each state q of the automaton. This coloring of the states $q \in Q$ of the automaton induces a coloring of the nodes of its run-trees, in the expected way. Following the principles of parity games, an infinite branch of such a run-tree is declared winning when the greatest color occurring infinitely often in it is even. A run-tree of the automaton is then accepted precisely when all its infinite branches are winning. In the sequel, we find convenient to consider the set $Col = \Omega(Q) \uplus \{\epsilon\}$ of colors appearing in the alternating parity automaton \mathcal{A} under study. The extra color ϵ is added as a neutral color, in order to reflect the comonadic nature of colors, as we will explain in the later Sect. 4. The following definition will also be useful in the sequel, in order to connect the alternating parity automaton \mathcal{A} and the finitary semantics of linear logic:

Definition 1. *Given a state $q \in Q$ and an n -ary constructor $a \in \Sigma$, we say that an n -tuple $\alpha \in (\mathcal{P}_{fin}(Col \times Q))^n$ satisfies the formula $\delta(q, a)$ when α is of the form*

$$\alpha = (\{ (c_{1i_1}, q_{1i_1}) \mid i_1 \in I_1 \} , \dots , \{ (c_{ni_n}, q_{ni_n}) \mid i_n \in I_n \})$$

and there exists an n -tuple of subsets $J_1 \subseteq I_1, \dots, J_n \subseteq I_n$ such that

$$\bigwedge_{k=1}^n \bigwedge_{j_k \in J_k} (k, q_{kj_k}) \tag{5}$$

defines a conjunctive clause of the formula $\delta(q, a)$, and such that moreover

$$\forall k \in \{1, \dots, n\} \quad \forall j \in J_k \quad c_{kj} = \Omega(q_{kj}).$$

In other words, α is an n -tuple of sets $\{(c_{1i_k}, q_{1i_k}) \mid i_k \in I_k\}$ of states annotated with colors, each of them corresponding to one of the n subtrees below the symbol a . Moreover, each such set should contain a subset $\{(\Omega(q_{1i_k}), q_{1i_k}) \mid i_k \in J_k\}$ of appropriately colored states, such that (5) defines a conjunctive clause of the formula $\delta(q, a)$. The general idea is that the n -tuple is allowed to contain more colored states than what is strictly required for the transition $\delta(q, a)$ to be performed by the alternating parity automaton \mathcal{A} . This definition will be crucial in the construction of the finitary semantics which, we will see, is based on downward-closed sets and subtyping.

4 The Scott Semantics of Linear Logic

Here, we adapt the infinitary and colored relational semantics of linear logic formulated in [4,5] to the finitary Scott semantics, where formulas of linear logic are interpreted as partial orders. The semantics of linear logic is *qualitative* in the technical sense that its exponential modality $!$ is interpreted using the finite *powerset* construction, which transports finite sets into finite sets, in contrast to the finite multiset construction used in the traditional and *quantitative* relational semantics. The terminology of Scott semantics comes from the fact that in the derived semantics of the simply-typed λ -calculus, every type is interpreted as a prime algebraic complete lattice, and every simply-typed λ -term as a Scott-continuous function. So, let **ScottL** denote the category with preorders $\mathbf{A} = (A, \leq_A)$ as objects and downward-closed binary relations $R \subseteq A \times B$ as morphisms $(A, \leq_A) \rightarrow (B, \leq_B)$. Here, by a downward-closed relation, we mean a binary relation R such that for all $a, a' \in A$ and $b, b' \in B$, one has :

$$(a, b) \in R \quad \text{and} \quad a \leq_A a' \quad \text{and} \quad b' \leq_B b \quad \Rightarrow \quad (a', b') \in R.$$

The binary relation R is thus downward closed in the partial order $(A, \leq_A)^{op} \times (B, \leq_B)$ interpreting the formula $(A, \leq_A) \multimap (B, \leq_B)$ in the Scott semantics. The intuition guiding this property is that if a binary relation R interpreting a proof of linear logic can produce an output b from an input a , then the same binary relation can also produce a less informative output b' from a more informative input a' . It is well-known in the literature on linear logic that this “saturation property” is essential in order to obtain a relational semantics of linear logic with a qualitative (that is, based on finite sets instead of finite multisets) interpretation of the exponential modality. This remark is generally attributed to Ehrhard, see [8] for details. The composition in **ScottL** is relational, since relational composition preserves the property of being downward-closed. The identity morphism over (A, \leq_A) is

$$id_A = \{(a', a) \mid a \leq_A a'\}$$

ScottL is a compact closed category with products, with

$$\begin{aligned} (A, \leq_A) \otimes (B, \leq_B) &= (A \times B, \leq_A \times \leq_B) & \mathbf{1} &= (\{\star\}, =) \\ (A, \leq_A) \&(B, \leq_B) &= (A \uplus B, \leq_A \uplus \leq_B) & \top &= (\emptyset, \emptyset) \\ (A, \leq_A)^\perp &= (A, \geq_A) \end{aligned}$$

The exponential modality

$$! : A \mapsto !A : \mathbf{ScottL} \longrightarrow \mathbf{ScottL}$$

is then defined by associating to the ordered set (A, \leq_A) the set $\mathcal{P}_{fin}(A)$ of finite subsets of A , where two finite subsets u and v are ordered in the following way:

$$u \leq_{!A} v \iff \forall a \in u, \exists b \in v, u \leq_A v.$$

Recall that the endofunctor $!$ transports every morphism $R : A \rightarrow B$ of the category **ScottL** to the following morphism:

$$!R = \{(u, v) \in !A \times !B \mid \forall b \in v \exists a \in u (a, b) \in R\} : !A \rightarrow !B$$

The endofunctor $!$ is in fact a comonad and defines a Seely category, and thus a model of full propositional linear logic, based on the category **ScottL**, see for instance [11].

The Coloring Comonad. As we have shown in [4,5], the treatment of colors by alternating parity automata follows essentially the same comonadic principles as the treatment of copies in linear logic. This connection between higher-order model checking and linear logic leads to a coloring monoidal comonad \square on the relational semantics of linear logic, which we adapt here to the qualitative Scott semantics. To that purpose, we fix a finite set of colors Col containing a neutral element ϵ , and consider the coloring function $Q \rightarrow Col$ which associates a color to every state of a parity tree automaton \mathcal{A} , see the previous discussion in Sect. 3. The modality \square is then defined in the following way for an ordered set (A, \leq_A) and a morphism $R : (A, \leq_A) \rightarrow (B, \leq_B)$:

$$\begin{aligned} \square (A, \leq_A) &= (A, \leq_A) \&\cdots \&(A, \leq_A) \\ &\cong (\{(c, a) \mid c \in Col, a \in A\}, \leq_{\square A}) \\ (c_1, a) \square R (c_2, b) &\text{ iff } c_1 = c_2 \text{ and } a R b \end{aligned}$$

where $(c_1, a) \leq_{\square A} (c_2, a')$ iff $c_1 = c_2$ and $a \leq_A a'$. The comonadic structure of \square is provided by the following structural morphisms

$$\begin{aligned} \mathbf{dig}_A &= \{((\max(c_1, c_2), a), (c_1, (c_2, a'))) \mid a' \leq_A a\} & : \square A \rightarrow \square \square A \\ \mathbf{der}_A &= \{((\epsilon, a), a') \mid a' \leq_A a\} & : \square A \rightarrow A \\ m_{A,B} &= \{(((c, a), (c, b)), ((c, (a', b')))) \mid a' \leq_A a, b' \leq_B b\} & : \square A \otimes \square B \rightarrow \square(A \otimes B) \\ m_1 &= \{(\star, (c, \star)) \mid c \in Col\} & : \mathbf{1} \rightarrow \square \mathbf{1} \end{aligned}$$

As we did in the case of the relational semantics [4, 5], we define a distributive law $\lambda : ! \circ \square \Rightarrow \square \circ !$ between the comonads $!$ and \square defined as the natural transformation:

$$\lambda_A = \{ (\{ (c_j, a'_j) \}, (c, \{ a_i \})) \mid \forall i \exists j \ c = c_j \text{ and } a_i \leq_A a'_j \} : !\square A \rightarrow \square! A$$

The existence of such a distributive law λ enables us to equip the composite functor $\mathbf{!} = ! \circ \square$ with a comonadic structure. It appears moreover that this colored exponential functor $\mathbf{!}$ satisfies the axioms of a Seely category, and thus defines a model of full propositional linear logic. We denote by $\mathbf{ScottL}_{\mathbf{!}}$ its Kleisli category.

5 A Finitary Interpretation of the Simply-Typed λ -calculus

In order to simplify the discussion, we suppose given an alternating parity tree \mathcal{A} over a signature Σ , with set of states Q and with transition function δ . As a Kleisli category associated to a model of linear logic, the category $\mathbf{ScottL}_{\mathbf{!}}$ is cartesian closed and thus a model of the simply-typed λ -calculus. The simple types are interpreted inductively as

$$\llbracket \sigma \rightarrow \tau \rrbracket = \mathbf{!} \llbracket \sigma \rrbracket \multimap \llbracket \tau \rrbracket \quad \text{and} \quad \llbracket o \rrbracket = \perp = (Q, =)$$

The interpretation of the simply-typed λ -terms is standard, except for the interpretation of the elements of the ranked alphabet Σ , seen as here constants of the simply-typed λ -calculus, which are interpreted as follows:

$$\llbracket a \rrbracket_{\mathcal{A}} = \{ (\alpha, q) \mid q \in Q \text{ and } \alpha \text{ satisfies the formula } \delta(q, a) \}$$

As explained in [5] in the case of the quantitative relational semantics of linear logic, this interpretation of the elements of Σ corresponds to a Church encoding of the alternating parity automaton \mathcal{A} , encoded in the present case in the qualitative Scott semantics of linear logic.

Example 2. Recall the two transitions (3) introduced as running example in Sect. 3:

$$\delta(q_0, \mathbf{if}) = (2, q_0) \wedge (2, q_1) \quad \delta(q_1, \mathbf{if}) = (1, q_1) \wedge (2, q_0)$$

Setting $c_i = \Omega(q_i)$, these transitions imply that

$$(u_1, u_2, q_0) \in \llbracket \mathbf{if} \rrbracket_{\mathcal{A}} \quad \text{and} \quad (v_1, v_2, q_1) \in \llbracket \mathbf{if} \rrbracket_{\mathcal{A}}$$

for all finite sets $u_1, u_2, v_1, v_2 \in \mathbf{!} \perp = \mathcal{P}_{fin}(Col \times Q)$ satisfying moreover that $\{(c_0, q_0), (c_1, q_1)\} \subseteq u_2$, that $(c_1, q_1) \in v_1$ and that $(c_0, q_0) \in v_2$.

Using these interpretations in \mathbf{ScottL} of the elements of the ranked alphabet Σ , we construct the interpretation

$$\llbracket \Gamma \vdash t :: \tau \rrbracket_{\mathcal{A}} \subseteq (\mathbf{!} \llbracket \sigma_1 \rrbracket \otimes \cdots \otimes \mathbf{!} \llbracket \sigma_n \rrbracket) \multimap \llbracket \tau \rrbracket$$

$$\frac{}{q \leq_{\perp} q} \quad \frac{\forall (c, \alpha) \in u \quad \exists (c, \beta) \in v \quad \alpha \leq_A \beta}{u \leq_{\sharp A} v} \quad \frac{v \leq_{\sharp A} u \quad \alpha \leq_B \beta}{u \rightarrow \alpha \leq_{\sharp A \rightarrow B} v \rightarrow \beta}$$

Fig. 3. Inference rules for the preorders associated with simple types.

$$\text{Ax} \quad \frac{\exists (\epsilon, \alpha') \in u \quad \alpha \leq_{[\sigma]} \alpha'}{x : u :: \sigma \vdash x : \alpha :: \sigma} \quad \frac{\Gamma, x : u :: \sigma \vdash M : \alpha :: \tau}{\Gamma \vdash \lambda x. M : u \rightarrow \alpha :: \sigma \rightarrow \tau} \quad \lambda$$

$$\text{App} \quad \frac{\Gamma_0 \vdash M : \{(c_1, \beta_1), \dots, (c_n, \beta_n)\} \rightarrow \alpha :: \sigma \rightarrow \tau \quad \Gamma_i \vdash N : \beta_i :: \sigma \quad (\forall i)}{\Gamma_0 \cup \square_{c_1} \Gamma_1 \cup \dots \cup \square_{c_n} \Gamma_n \vdash M N : \alpha :: \tau}$$

$$\delta \quad \frac{q \in Q \text{ and } \alpha \text{ satisfies } \delta(q, a)}{\emptyset \vdash a : \alpha \rightarrow q :: \sigma}$$

Fig. 4. Type-theoretic computation of denotations in **ScottL_‡**

of any λ -term t of type τ in a context of typed variables Γ , with constants in the ranked alphabet Σ . An alternative way to describe this interpretation is to express it as an intersection type system with subtyping, in the style of Coppo, Dezani, Honsell and Longo [3] and more recently Terui [11] in the framework of linear logic. In this formulation, sequents are of the following form

$$\Gamma = x_1 : u_1 :: \sigma_1, \dots, x_n : u_n :: \sigma_n \vdash t : \alpha :: \tau$$

where $u_i \in \sharp[\sigma_i]$ and $\alpha \in [\tau]$. The typing rules are presented in Fig. 4, with the subtyping relation \leq_A defined inductively in Fig. 3. Note that the coloring $\square_c \Gamma$ of a context is defined inductively as

$$\begin{aligned} \square_c (x : u :: \sigma, \Gamma) &= x : \square_c u :: \sigma, \square_c \Gamma \\ \square_c \{(c_i, \alpha_i)\} &= \{(\max(c, c_i), \alpha_i)\} \end{aligned}$$

Proposition 3. *The sequent*

$$\Gamma = x_1 : u_1 :: \sigma_1, \dots, x_n : u_n :: \sigma_n \vdash t : \alpha :: \tau$$

is provable in this intersection type system if and only if

$$(u_1, \dots, u_n, \alpha) \in [\Gamma \vdash t :: \tau]_A \subseteq (\sharp[\sigma_1] \otimes \dots \otimes \sharp[\sigma_n]) \multimap [\tau]$$

6 The recursion operator **Y**

At this stage, we are ready to shift from the colored semantics of the simply-typed λ -calculus formulated in Sect. 5 to a colored semantics of the simply-typed λY -calculus. To that purpose, we construct a Conway operator **Y** in the

category **FinScottL** defined as the full subcategory of **ScottL** consisting of the *finite* ordered sets. Note that **FinScottL** defines a Seely category, and thus a model of full propositional linear logic. We will see in the next section that the finitary nature of the model will enable us to establish the *decidability* of the selection problem (Theorem 3). On the other hand, shifting from **ScottL** to **FinScottL** means that we cannot interpret any more infinitary types like the type of natural numbers. The Conway operator **Y** is defined a family of operations $\mathbf{Y}_{X,A}$ transporting a binary downward-closed relation

$$R : \Downarrow X \otimes \Downarrow A \multimap A$$

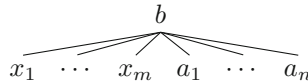
into a binary downward-closed relation

$$\mathbf{Y}_{X,A}(R) : \Downarrow X \multimap A$$

and satisfying a series of conditions originally stated by Bloom and Esik [1] in cartesian closed categories, and adapted in [4] to the particular framework of Seely categories. Note that such a Conway operator on **FinScottL** defines a Conway operator in the sense of [1] in the cartesian-closed category **FinScottL_↓**. Just as in the case of the relational semantics, see [4] for details, the important point here is that the colors added to the original Scott semantics will enable us to alternate least and greatest fixpoints (and thus inductive and coinductive reasoning) in the definition of the fixpoint operator **Y**, using the appropriate parity condition.

Semantic Run-trees. Given a relation $R : \Downarrow X \otimes \Downarrow A \multimap A$ and $a \in A$, we define the set **comp**(R, a) of *semantic run-trees* of R producing $a \in A$ as the set of possibly infinite $(X \uplus A)$ -labelled trees, with nodes colored by elements of Col , and such that the four conditions below are satisfied:

1. the root of the tree is labelled by a , and has neutral color ϵ ,
2. the inner nodes of the tree are labelled by elements of the set A ,
3. the leaves are labelled by elements of the set $X \uplus A$,
4. for every node labelled by an element $b \in A$:
 - if b is an inner node, letting a_1, \dots, a_n denote the labels of its children belonging to A and x_1, \dots, x_m the labels belonging to X :



and letting c_i (resp. d_j) be the color of the node labelled x_i (resp. a_j),

$$(\{(c_1, x_1), \dots, (c_m, x_m)\}, \{(d_1, a_1), \dots, (d_n, a_n)\}, b) \in R$$

- if b is a leaf, then $(\emptyset, \emptyset, b) \in R$.

At this point, we adapt to semantic run-trees the usual acceptance condition on the run-trees of an alternating parity automata: an infinite branch of the semantic run-tree is winning if and only if an element of $Col \setminus \{\epsilon\}$ occurs infinitely often along it, and if the maximal such element is even. A semantic run-tree is declared winning if and only if all its infinite branches are.

Given a semantic run-tree *witness*, we define the set $\mathbf{leaves}(witness) \subseteq \downarrow X$ as the set of elements (c, x) where (c', x) is a leaf of *witness* labelled with $x \in X$, and c is the maximal color encountered on the path from the leaf to the root of *witness*.

Fixpoint Operator. We now define the fixpoint of a binary relation

$$R : \downarrow X \otimes \downarrow A \multimap A$$

as the downward-closed binary relation

$$\mathbf{Y}_{X,A}(R) = \{ (u, a) \mid \exists witness \in \mathbf{comp}(R, a) \text{ with } u = \mathbf{leaves}(witness) \text{ and } witness \text{ is a winning semantic run-tree.} \} \quad (6)$$

Proposition 4. *The fixpoint operator \mathbf{Y} is a Conway operator over $\mathbf{FinScottL}$. The Kleisli category $\mathbf{FinScottL}_\downarrow$ of \downarrow is therefore a model of the $\lambda\mathbf{Y}$ -calculus.*

As in Sect. 5, we find useful and even illuminating to formulate a type-theoretic counterpart to our definition of the Conway operator $\mathbf{Y}_{X,A}$ provided by the following typing rule Y_σ which should be added to the type system of Fig. 4 :

$$Y_\sigma \frac{\Gamma_0 \vdash M : \{(c_1, \beta_1), \dots, (c_n, \beta_n)\} \rightarrow \alpha :: \sigma \rightarrow \sigma \quad \Gamma_i \vdash Y_\sigma M : \beta_i :: \sigma}{\Gamma_0 \sqcup \square_{c_1} \Gamma_1 \cup \dots \cup \square_{c_n} \Gamma_n \vdash Y_\sigma M : \alpha :: \sigma}$$

In the resulting intersection type system, derivations of infinite depth are allowed, and have colored nodes, defined as follows:

- for every occurrence of the rule Y_σ , we assign color c_i to the node $\Gamma_i \vdash Y_\sigma M : \beta_i :: \sigma$.
- all the other nodes are assigned the neutral color ϵ .

An infinite derivation tree is then accepted when all its branches are winning, in the same sense as for the branches of a semantic run-tree.

Theorem 1. *Given a $\lambda\mathbf{Y}$ -term t , the sequent*

$$\Gamma = x_1 : u_1 :: \sigma_1, \dots, x_n : u_n :: \sigma_n \vdash t : \alpha :: \tau$$

has a winning derivation tree in the type system with fixpoints iff

$$(u_1, \dots, u_n, \alpha) \in \llbracket \Gamma \vdash t :: \tau \rrbracket_A \subseteq (\downarrow \llbracket \sigma_1 \rrbracket \otimes \dots \otimes \downarrow \llbracket \sigma_n \rrbracket) \multimap \llbracket \tau \rrbracket$$

At this point, we take advantage of the correspondence recalled in Proposition 1 between higher-order recursion schemes (HORS) on the ranked alphabet Σ , and closed λY -terms with constants in the same alphabet Σ . Indeed, the correspondence enables us to justify the following typing rule for HORS :

$$\frac{\Gamma_0, F : \{(c_1, \beta_1), \dots, (c_n, \beta_n)\} :: \sigma \vdash \mathcal{R}(F) : \alpha :: \sigma \quad \Gamma_i \vdash F : \beta_i :: \sigma \quad (\forall i)}{\Gamma_0 \cup \square_{c_1} \Gamma_1 \cup \dots \cup \square_{c_n} \Gamma_n \vdash F : \alpha :: \sigma}$$

which provides a direct mean to type the HORS \mathcal{G} in the intersection type system, in such a way as to reflect its interpretation $\llbracket \mathcal{G} \rrbracket_{\mathcal{A}} \subseteq Q$ in the Scott semantics.

7 Decidability of the Selection Problem

The first theorem of the section establishes a perfect correspondence between our finitary interpretation $\llbracket \mathcal{G} \rrbracket_{\mathcal{A}}$ of the higher-order recursion scheme \mathcal{G} in the Scott semantics, and the set of accepting states of the automaton \mathcal{A} :

Theorem 2. *An alternating parity tree automaton \mathcal{A} has an accepting run-tree with initial state q_0 over the value tree $\langle \mathcal{G} \rangle$ of a higher-order recursion scheme \mathcal{G} if and only if $q_0 \in \llbracket \mathcal{G} \rrbracket_{\mathcal{A}}$.*

By Theorem 1, checking whether $q_0 \in \llbracket \mathcal{G} \rrbracket_{\mathcal{A}}$ is equivalent to checking whether there exists a derivation of the sequent $\emptyset \vdash S : q_0 :: o$ in the colored intersection type system defined in Sect. 6. Since the interpretation of simple types in **FinScottL** is finite, only finitely many intersection types and contexts may occur in such a derivation. Hence, searching for a derivation of the sequent $\emptyset \vdash S : q_0 :: o$ reduces in this case to solving a finite parity game whose nodes are precisely the sequents of the derivation tree. This has the following immediate consequence:

Corollary 1. *The local model-checking problem is decidable.*

Recall moreover that the existence of a winning strategy in a finite parity game implies that there exists a *memoryless* winning strategy. In this setting, winning strategies correspond to winning derivation trees of the intersection type system, and memoryless strategies correspond to derivation trees admitting a finite representation using backtracking pointers. From such a finite representation π , one can define a higher-order recursion scheme \mathcal{G}_q on a ranked alphabet $\Sigma_{\mathcal{A}}$ obtained from Σ by annotating every terminal a with elements of its interpretation $\llbracket a \rrbracket_{\mathcal{A}}$. The HORS \mathcal{G}_q has a non-terminal $F_{\alpha}(o)$ for every occurrence o of the non-terminal F in the finite representation π of the derivation tree, where α is the intersection type of the occurrence o of F in π . Each occurrence o of a non-terminal F then induces a rewrite rule $F_{\alpha}(o) \rightarrow_{\mathcal{G}_q} \text{term}(o)$ where $\text{term}(o)$ is an annotated version of the λ -term $\mathcal{R}(F)$ coming from the original scheme \mathcal{G} . The annotation of $\text{term}(o)$ is obtained by annotating the non-terminals and the

terminals of $\mathcal{R}(F)$ with the intersection types occurring in the finite representation π of the derivation tree. This defines a higher-order recursion scheme \mathcal{G}_q , which generates a run-tree $\langle \mathcal{G}_q \rangle$ of the alternating parity tree automaton \mathcal{A} over $\langle \mathcal{G} \rangle$. As a consequence:

Theorem 3. *The selection problem is decidable.*

8 Conclusions and Perspectives

In this paper, we explain how to apply our semantic approach to higher-order model-checking based on linear logic, in order to establish the decidability of local model-checking and of the selection problem. Our approach provides a rigorous and compositional approach to higher-order model-checking, and adapts to the inductive-coinductive framework of MSO logic a nice and well-established connection between linear logic, Scott domains, and intersection types. Future work includes a detailed comparison with a similar line of work on finite models of the λY -calculus currently developed by Salvati and Walukiewicz [10].

References

1. Bloom, S.L., Ésik, Z.: Fixed-point operations on CCC's. Part I. Theor. Comput. Sci. **155**(1), 1–38 (1996)
2. Carayol, A., Serre, O.: Collapsible pushdown automata and labeled recursion schemes: equivalence, safety and effective selection. In: LICS 2012. pp. 165–174. IEEE Computer Society (2012)
3. Coppo, M., Dezani-Ciancaglini, M., Honsell, F., Longo, G.: Extended Type Structures and Filter Lambda Models. In: Lolli, G., Longo, G., Marcja, A. (eds.) Logic Colloquium 82, pp. 241–262. North-Holland, Amsterdam (1984)
4. Grellois, C., Melliès, P.-A.: An infinitary model of linear logic. In: Pitts, A. (ed.) FOSSACS 2015. LNCS, vol. 9034, pp. 41–55. Springer, Heidelberg (2015)
5. Grellois, C., Melliès, P.A.: Relational semantics of linear logic and higher-order model-checking. <http://arxiv.org/abs/1501.04789>. (accepted at CSL 2015)
6. Haddad, A.: Shape-preserving transformations of higher-order recursion schemes. Ph.D. thesis, Université Paris Diderot (2013)
7. Knapik, T., Niwiński, 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)
8. Melliès, P.A.: Categorical semantics of linear logic. In: Interactive Models of Computation and Program Behaviour, Panoramas et Synthèses 27. pp. 1–196. Soci Mathématique de France (2009)
9. Ong, C.H.L.: On model-checking trees generated by higher-order recursion schemes. In: LICS, pp. 81–90 (2006)
10. Salvati, S., Walukiewicz, I.: A model for behavioural properties of higher-order programs (2015). <https://hal.archives-ouvertes.fr/hal-01145494>
11. Terui, K.: Semantic evaluation, intersection types and complexity of simply typed lambda calculus. In: Tiwari, A. (ed.) RTA 2012. LIPIcs, vol. 15, pp. 323–338. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Wadern (2012)

Complexity of Propositional Independence and Inclusion Logic

Miika Hannula¹, Juha Kontinen¹, Jonni Virtema^{2(✉)}, and Heribert Vollmer²

¹ University of Helsinki, Department of Mathematics and Statistics, Helsinki, Finland
`{miika.hannula, juha.kontinen}@helsinki.fi`

² Leibniz Universität Hannover, Institut für Theoretische Informatik,
Fakultät für Elektrotechnik und Informatik, Hanover, Germany
`jonni.virtema@uta.fi, vollmer@thi.uni-hannover.de`

Abstract. We classify the computational complexity of the satisfiability, validity and model-checking problems for propositional independence and inclusion logic and their extensions by the classical negation.

Keywords: Propositional logic · Team semantics · Dependence · Independence · Inclusion · Satisfiability · Validity · Model-checking

1 Introduction

Dependence logic [17] is a new logical framework for formalising and studying various notions of dependence and independence that are important in many scientific disciplines such as experimental physics, social choice theory, computer science, and cryptography. Dependence logic extends first-order logic by dependence atoms

$$\text{dep}(x_1, \dots, x_n, y) \tag{1}$$

expressing that the value of the variable y is functionally determined on the values of x_1, \dots, x_n . Satisfaction for formulas of dependence logic is defined using sets of assignments (*teams*) and not in terms of single assignments as in first-order logic. Whereas dependence logic studies the notion of functional dependence, independence and inclusion logic (introduced in [6] and [5], respectively) formalize the concepts of independence and inclusion. Independence logic (inclusion logic) is obtained from dependence logic by replacing dependence atoms by the so-called independence atoms $\mathbf{x} \perp_{\mathbf{y}} \mathbf{z}$ (inclusion atoms $\mathbf{x} \subseteq_{\mathbf{y}} \mathbf{z}$). The intuitive meaning of the independence atom is that the variables of the tuples \mathbf{x} and \mathbf{z} are independent of each other for any fixed value of the variables in \mathbf{y} , whereas the inclusion atom declares that all values of the tuple \mathbf{x} appear also as values of \mathbf{y} . In database theory these atoms correspond to the so-called embedded multivalued dependencies and inclusion dependencies (see, e.g., [7]). Independence atoms have also a close connection to conditional independence in statistics.

The first and the second author was supported by the Academy of Finland grants 264917 and 275241. The third author was supported by a grant from the Jenny and Antti Wihuri Foundation.

The topic of this article is propositional team semantics which has received relatively little attention so far. On the other hand, modal team semantics has been studied actively. Since the propositional logics studied in this article are fragments of the corresponding modal logics, some upper bounds trivially transfer to the propositional setting. The study of propositional team semantics as a subject of independent interest was initiated after surprising connections between propositional team semantics and the so-called *inquisitive semantics* was discovered (see [19] for details). The first systematic study on the expressive power of propositional dependence logic and many of its variants is due to [19, 20]. In the same works natural deduction type inference systems for these logics are also developed, whereas in [16] a complete Hilbert-style axiomatization for propositional dependence logic is presented.

The computational aspects of (first-order) dependence logic and its variants have been actively studied, and are now quite well understood. On the other hand, the complexity of the propositional versions of these logics have not been systematically studied except for [18] in which the validity problem of propositional dependence logic was shown to be NEXPTIME-complete. In this article we study the complexity of satisfiability, validity and model-checking of propositional independence and inclusion logic and their extensions by the classical negation. The classical negation has turned out to be a very powerful connective in the settings of first-order and modal team semantics, see e.g., [11] and [12]. Our results (see Table 1) show that the same is true in the propositional setting. In particular, our main result shows that the satisfiability and validity problems of the extensions of propositional independence and inclusion logic by the classical negation are complete for alternating exponential time with polynomially many alternations (AEXPTIME(poly)).

2 Preliminaries

In this section we define the basic concepts and results relevant to team-based propositional logics. We assume that the reader is familiar with propositional logic.

2.1 Syntax and Semantics

Let D be a finite, possibly empty, set of proposition symbols. A function $s : D \rightarrow \{0, 1\}$ is called an *assignment*. A set X of assignments $s : D \rightarrow \{0, 1\}$ is called a *team*. The set D is the *domain* of X . We denote by 2^D the set of *all assignments* $s : D \rightarrow \{0, 1\}$.

Let Φ be a set of proposition symbols. The syntax for propositional logic $\text{PL}(\Phi)$ is defined as follows.

$$\varphi ::= p \mid \neg p \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi), \quad \text{where } p \in \Phi.$$

We write $\text{Var}(\varphi)$ for the set of all proposition symbols that appear in φ . We denote by \models_{PL} the ordinary satisfaction relation of propositional logic defined via assignments in the standard way. Next we give team semantics for propositional logic.

Table 1. Overview of the results (completeness results if not stated otherwise, \sim refers to classical negation).

	SAT	VAL	MC
$\text{PL}[\perp_c]$	NP	in $\text{coNEXPTIME}^{\text{NP}}$	NP
$\text{PL}[\subseteq]$	EXPTIME [10]	coNP	in P [9]
$\text{PL}[\perp_c, \sim], \text{PL}[\subseteq, \sim]$	AEXPTIME(poly)	AEXPTIME(poly)	PSPACE

Definition 1. Let Φ be a set of atomic propositions and let X be a team. The satisfaction relation $X \models \varphi$ is defined as follows.

$$\begin{aligned}
 X \models p &\Leftrightarrow \forall s \in X : s(p) = 1. \\
 X \models \neg p &\Leftrightarrow \forall s \in X : s(p) = 0. \\
 X \models (\varphi \wedge \psi) &\Leftrightarrow X \models \varphi \text{ and } X \models \psi. \\
 X \models (\varphi \vee \psi) &\Leftrightarrow Y \models \varphi \text{ and } Z \models \psi, \text{ for some } Y, Z \text{ such that } Y \cup Z = X.
 \end{aligned}$$

Note that in team semantics \neg is not the classical negation (denoted by \sim in this article) but a so-called *dual* negation that does not satisfy the law of excluded middle. Next proposition shows that the team semantics and the ordinary semantics for propositional logic defined via assignments coincide.

Proposition 1 ([17]). Let φ be a formula of propositional logic and let X be a propositional team. Then $X \models \varphi$ iff $\forall s \in X : s \models_{PL} \varphi$.

The syntax of *propositional dependence logic* $\text{PD}(\Phi)$ is obtained by extending the syntax of $\text{PL}(\Phi)$ by

$$\varphi ::= \text{dep}(p_1, \dots, p_n, q), \quad \text{where } p_1, \dots, p_n, q \in \Phi.$$

The semantics for the propositional dependence atoms are defined as follows:

$$\begin{aligned}
 X \models \text{dep}(p_1, \dots, p_n, q) &\Leftrightarrow \forall s, t \in X : s(p_1) = t(p_1), \dots, s(p_n) = t(p_n) \\
 &\text{implies that } s(q) = t(q).
 \end{aligned}$$

The next proposition is very useful when determining the complexity of PD, and it is proved analogously as for first-order dependence logic [17].

Proposition 2 (Downwards Closure). Let φ be a PD-formula and let $Y \subseteq X$ be propositional teams. Then $X \models \varphi$ implies $Y \models \varphi$.

In this article we study the variants of PD obtained by replacing dependence atoms in terms of the so-called independence or inclusion atoms: The syntax of *propositional independence logic* $\text{PL}[\perp_c](\Phi)$ is obtained by extending the syntax of $\text{PL}(\Phi)$ by the grammar rule

$$\varphi ::= \mathbf{q} \perp_{\mathbf{p}} \mathbf{r},$$

where \mathbf{p} , \mathbf{q} , and \mathbf{r} are finite tuples of proposition variables (not necessarily of the same length). The syntax of *propositional inclusion logic* $PL[\sqsubseteq](\Phi)$ is obtained by extending the syntax of $PL(\Phi)$ by the grammar rule

$$\varphi ::= \mathbf{p} \sqsubseteq \mathbf{q},$$

where \mathbf{p} and \mathbf{q} are finite tuples of proposition variables with the same length. Satisfaction for these atoms is defined as follows. If $\mathbf{p} = (p_1, \dots, p_n)$ and s is an assignment, we write $s(\mathbf{p})$ for $(s(p_1), \dots, s(p_n))$.

$$\begin{aligned} X \models \mathbf{q} \perp_{\mathbf{p}} \mathbf{r} &\Leftrightarrow \forall s, t \in X : \text{if } s(\mathbf{p}) = t(\mathbf{p}) \\ &\quad \text{then there exists } u \in X : u(\mathbf{pq}) = s(\mathbf{pq}) \text{ and } u(\mathbf{r}) = t(\mathbf{r}). \\ X \models \mathbf{p} \sqsubseteq \mathbf{q} &\Leftrightarrow \forall s \in X \exists t \in X : s(\mathbf{p}) = t(\mathbf{q}). \end{aligned}$$

It is easy to check that neither $PL[\perp_c]$ nor $PL[\sqsubseteq]$ is a downward closed logic (cf. Proposition 2). However, analogously to first-order inclusion logic [5], the formulas of $PL[\sqsubseteq]$ have the following closure property.

Proposition 3 (Closure Under Unions). *Let $\varphi \in PL[\sqsubseteq]$ and let X_i , for $i \in I$, be teams. Suppose that $X_i \models \varphi$, for each $i \in I$. Then $\bigcup_{i \in I} X_i \models \varphi$.*

We will also consider the extensions of PL , $PL[\perp_c]$ and $PL[\sqsubseteq]$, by the classical negation \sim with the standard semantics:

$$X \models \sim\varphi \Leftrightarrow X \not\models \varphi.$$

These extensions are denoted by $PL[\sim]$, $PL[\perp_c, \sim]$ and $PL[\sqsubseteq, \sim]$, respectively.

2.2 Auxiliary Operators

The following additional operators will be used in this paper:

$$\begin{aligned} X \models \varphi \circledast \psi &\Leftrightarrow X \models \varphi \text{ or } X \models \psi, \\ X \models \varphi \otimes \psi &\Leftrightarrow \forall Y, Z \subseteq X : \text{if } Y \cup Z = X, \text{ then } Y \models \varphi \text{ or } Z \models \psi, \\ X \models \varphi \multimap \psi &\Leftrightarrow \forall Y \subseteq X : \text{if } Y \models \varphi, \text{ then } Y \models \psi, \\ X \models \max(x_1, \dots, x_n) &\Leftrightarrow \{(s(x_1), \dots, s(x_n)) \mid s \in X\} = \{0, 1\}^n. \end{aligned}$$

If $X \models \max(\mathbf{x})$, we say that X is *maximal over \mathbf{x}* . If tuples \mathbf{x} and \mathbf{y} are pairwise disjoint and $X \models \max(\mathbf{x}) \wedge \mathbf{x} \perp \mathbf{y}$, then we say that X is *maximal over \mathbf{x}* for all \mathbf{y} . For the proof of the following proposition, see Appendix of the arXiv version of the paper ([8]).

Proposition 4. *$\text{dep}(\cdot)$, \circledast , \otimes , \multimap , and $\max(\cdot)$ translate into $PL[\sim]$ in polynomial time.*

Table 2. Complexity of satisfiability, validity, and model checking of PL and PD. All results are completeness results.

	SAT	VAL	MC	References
PL	NP	coNP	NC ¹	[1, 3, 13]
PD	NP	NEXPTIME	NP	[4, 14, 18]

2.3 Satisfiability, Validity, and Model Checking in Team Semantics

Next we define satisfiability and validity in the context of team semantics. Let L be a logic with team semantics. A formula $\varphi \in L$ is *satisfiable*, if there exists a non-empty team X such that $X \models \varphi$. A formula $\varphi \in L$ is *valid*, if $X \models \varphi$ holds for every non-empty team X such that the proposition symbols that occur in φ are in the domain of X .¹ Note that when the team is empty, satisfaction becomes easy to decide (see Proposition 6 in Appendix of the arXiv version [8]).

The satisfiability problem $\text{SAT}(L)$ and the validity problem $\text{VAL}(L)$ are then defined in the obvious manner: Given a formula $\varphi \in L$, decide whether the formula is satisfiable (valid, respectively). The variant of the model checking problem that we are concerned in this article is the following: Given a formula $\varphi \in L$ and a team X , decide whether $X \models \varphi$. See Table 2 for known complexity results on PL and PD.

3 Complexity of Satisfiability and Validity

In this section we consider the complexity of the satisfiability and validity problems for propositional independence logic and inclusion logic, and their extensions by the classical negation \sim .

3.1 The Logics $\text{PL}[\perp_c]$ and $\text{PL}[\sqsubseteq]$

We consider first the complexity of satisfiability and validity for propositional independence logic. The following simple lemma turns out to be very useful.

Lemma 1. *Let $\varphi \in (\text{PL}[\perp_c])$ and X a team such that $X \models \varphi$. Then $\{s\} \models \varphi$, for all $s \in X$.*

Proof. The claim is proved using induction on the construction of φ . It is easy to check that a singleton team satisfies all independence atoms, and the cases corresponding to disjunction and conjunction are straightforward. \square

Theorem 1. *$\text{SAT}(\text{PL}[\perp_c])$ is complete for NP.*

¹ It is easy to show that all of the logics considered in this article have the so-called locality property, i.e., satisfaction of a formula depends only on the values of the proposition symbols that occur in the formula [5].

Proof. Note first that since $\text{SAT}(\text{PL})$ is NP-complete, it follows by Proposition 1 that $\text{SAT}(\text{PL}[\perp_c])$ is NP-hard. For containment in NP, note that by Lemma 1, a formula $\varphi \in \text{PL}[\perp_c]$ is satisfiable iff it is satisfied by some singleton team $\{s\}$. It is immediate that for any s , $\{s\} \models \varphi$ iff $\{s\} \models \varphi^T$, where $\varphi^T \in \text{PL}$ is acquired from φ by replacing all independence atoms by $(p \vee \neg p)$. Thus it follows that φ is satisfiable iff φ^T is satisfiable. Therefore, the claim follows. \square

Theorem 2. $\text{VAL}(\text{PL}[\perp_c])$ is hard for NEXPTIME and is in $\text{coNEXPTIME}^{\text{NP}}$.

Proof. Since the dependence atom $\text{dep}(x, y)$ is equivalent to the independence atom $y \perp_x y$ and $\text{VAL}(\text{PD})$ is NEXPTIME-complete [18], hardness for NEXPTIME follows. We will show in Theorem 9 on p. 10 that the model checking problem for $\text{PL}[\perp_c]$ is complete for NP. It then follows that the complement of the problem $\text{VAL}(\text{PL}[\perp_c])$ is in $\text{NEXPTIME}^{\text{NP}}$: the question whether φ is in the complement of $\text{VAL}(\text{PL}[\perp_c])$ can be decided by guessing a subset X of 2^D , where D contains the set of proposition variables appearing in φ , and checking whether $X \not\models \varphi$. Therefore $\text{VAL}(\text{PL}[\perp_c]) \in \text{coNEXPTIME}^{\text{NP}}$. \square

Next we turn to propositional inclusion logic.

Theorem 3. ([10]). $\text{SAT}(\text{PL}[\subseteq])$ is complete for EXPTIME.

We end this section by determining the complexity of $\text{VAL}(\text{PL}[\subseteq])$.

Theorem 4. $\text{VAL}(\text{PL}[\subseteq])$ is complete for coNP.

Proof. Recall that PL is a sub-logic of $\text{PL}[\subseteq]$, and hence $\text{VAL}(\text{PL}[\subseteq])$ is hard for coNP. Therefore, it suffices to show $\text{VAL}(\text{PL}[\subseteq]) \in \text{coNP}$. It is easy to check that, by Proposition 3, a formula $\varphi \in \text{PL}[\subseteq]$ is valid iff it is satisfied by all singleton teams $\{s\}$. Note also that, over a singleton team $\{s\}$, an inclusion atom $(p_1, \dots, p_n) \subseteq (q_1, \dots, q_n)$ is equivalent to the PL-formula

$$\bigwedge_{1 \leq i \leq n} (p_i \wedge q_i) \vee (\neg p_i \wedge \neg q_i).$$

Denote by φ^* the PL-formula acquired by replacing all inclusion atoms in φ by their PL-translations. By the above, φ is valid iff φ^* is valid. Since $\text{VAL}(\text{PL})$ is in coNP the claim follows. \square

3.2 The Logics $\text{PL}[\perp_c, \sim]$ and $\text{PL}[\subseteq, \sim]$

Next we incorporate classical negation in our logics. The main result of this section is that the satisfiability and validity problems for $\text{PL}[\perp_c, \sim]$ and $\text{PL}[\subseteq, \sim]$ are complete for $\text{AEXPTIME}(\text{poly})$. The upper bound follows by an exponential-time alternating algorithm where alternation is bounded by formula depth. For the lower bound we first relate $\text{AEXPTIME}(\text{poly})$ to polynomial-time alternating Turing machines that query to oracles obtained from a quantifier prefix of polynomial length. We then show how to capture this characterization with our logics.

First we observe that classical negation gives rise to polynomial-time reductions between validity and satisfiability problems. Hence, we restrict attention to satisfiability hereafter.

Proposition 5. *Let $\varphi \in \text{PL}[\mathcal{C}, \sim]$ where $\mathcal{C} \subseteq \{\text{dep}(\cdot), \perp_c, \subseteq\}$. Then one can construct in polynomial time formulae $\psi, \theta \in \text{PL}[\mathcal{C}, \sim]$ such that*

- (i) φ is satisfiable iff ψ is valid, and
- (ii) φ is valid iff θ is satisfiable.

Proof. We define

$$\begin{aligned} \psi &:= \max(\mathbf{x}) \multimap ((p \vee \neg p) \vee (\varphi \wedge \sim(p \wedge \neg p))), \\ \theta &:= \max(\mathbf{x}) \wedge (\sim(p \wedge \neg p) \multimap \varphi), \end{aligned}$$

where \mathbf{x} lists $\text{Var}(\varphi)$. Note that $X \models \sim(p \wedge \neg p)$ iff X is non-empty. It is straightforward to show that (i) and (ii) hold. Also by Proposition 4, ψ and θ can be constructed in polynomial time from φ . \square

First we show the upper bound for $\text{SAT}(\text{PL}[\perp_c, \subseteq, \sim])$.

Theorem 5. $\text{SAT}(\text{PL}[\perp_c, \subseteq, \sim]) \in \text{AEXPTIME}(\text{poly})$.

Proof. First construct an APTIME algorithm for $\text{MC}(\text{PL}[\perp_c, \subseteq, \sim])$ such that its alternation is bounded by the size of φ . Such an algorithm is presented in Appendix of the arXiv version ([8]) and it considers triples (T, φ, I) where T is a team, φ a formula, and $I \in \{0, 1\}$, and it alternates if φ is either a conjunction or a disjunction. Then, given $\varphi \in \text{PL}[\perp_c, \subseteq, \sim]$, it suffices to existentially guess a possibly exponential-size team T with domain $\text{Var}(\varphi)$, and then implement the algorithm on $(T, \varphi, 1)$. \square

Let us then turn to the lower bound. We show that the satisfiability problems of $\text{PL}[\perp_c, \sim]$ and $\text{PL}[\subseteq, \sim]$ are both hard for $\text{AEXPTIME}(\text{poly})$. For this, we first relate $\text{AEXPTIME}(\text{poly})$ to polynomial-time oracle Turing machines. This approach is originally due to Orponen in [15], where the classes Σ_k^{EXP} and Π_k^{EXP} of the exponential-time hierarchy were characterized by polynomial-time constant-alternation oracle Turing machines that query to k oracles. Recall that the exponential-time hierarchy corresponds to the class of problems that can be recognized by an exponential-time alternating Turing machine with constantly many alternations. In the next theorem we generalize Orponen’s characterization to exponential-time alternating Turing machines with *polynomially* many alternations (i.e. the class $\text{AEXPTIME}(\text{poly})$) by allowing queries to polynomially many oracles.

By (A_1, \dots, A_k) we denote an efficient disjoint union of sets A_1, \dots, A_k , e.g. $(A_1, \dots, A_k) = \{(i, x) : x \in A_i, 1 \leq i \leq k\}$.

Theorem 6. *A set A belongs to the class $\text{AEXPTIME}(\text{poly})$ iff there exist a polynomial f and a polynomial-time alternating oracle Turing machine M such that, for all x ,*

$$x \in A \text{ iff } Q_1 A_1 \dots Q_{f(n)} A_{f(n)} (M \text{ accepts } x \text{ with oracles } (A_1, \dots, A_{f(n)})),$$

where n is the length of x and $Q_1, \dots, Q_{f(n)}$ alternate between \exists and \forall , i.e., $Q_{i+1} \in \{\forall, \exists\} \setminus \{Q_i\}$.

Proof. The proof is a straightforward generalization of the proof of Theorem 5.2. in [15] (See Appendix of the arXiv version [8]). \square

Using this theorem we now prove Theorem 7. For the quantification over oracles A_i , we use repetitively \forall and \sim . For simulating the computation of an alternating polynomial-time oracle Turing machine, we first quantify over polynomially many boolean sequences of polynomial length and then simulate the computation of a deterministic polynomial-time Turing machine which queries to the quantified oracles.

Theorem 7. $\text{SAT}(\text{PL}[\perp_c, \sim])$ and $\text{SAT}(\text{PL}[\subseteq, \sim])$ are hard for $\text{AEXPTIME}(\text{poly})$.

Proof. Let $A \in \text{AEXPTIME}(\text{poly})$. From Theorem 6 we obtain a polynomial f and an alternating oracle Turing machine M with running time bounded by g . By [2], the alternating machine can be replaced by a sequence of word quantifiers over a deterministic Turing machine. (Strictly speaking, [2] speaks only about a bounded number of alternations, but the generalization to the unbounded case is straightforward.) W.l.o.g. we may assume that each configuration of M has at most two configurations reachable in one step. It then follows by Theorem 6 that one can construct a polynomial-time deterministic oracle Turing machine M^* such that $x \in A$ iff

$$Q_1 A_1 \dots Q_{f(n)} A_{f(n)} Q'_1 \mathbf{y}_1 \dots Q'_{g(n)} \mathbf{y}_{g(n)} \\ (M^* \text{ accepts } (x, \mathbf{y}_1, \dots, \mathbf{y}_{g(n)}) \text{ with oracles } (A_1, \dots, A_{f(n)})),$$

where $Q_1, \dots, Q_{f(n)}$ and $Q'_1, \dots, Q'_{g(n)}$ are alternating sequences of quantifiers \exists and \forall , and each \mathbf{y}_i is a $g(n)$ -ary sequence of propositional variables where n is the length of x . Note that M^* runs in polynomial time also with respect to n . Using this characterization we now show how to reduce in polynomial time any x to a formula φ in $\text{PL}[\perp_c, \sim]$ (or in $\text{PL}[\subseteq, \sim]$) such that $x \in A$ iff φ is satisfiable. We construct φ inductively. As a first step, we let

$$\varphi := \max(\mathbf{q}\mathbf{r}\mathbf{y}) \wedge p_t \wedge \neg p_f \wedge \varphi_1$$

where

- \mathbf{q} and \mathbf{r} list propositional variables that are used for encoding oracles;
- \mathbf{y} lists propositional variables that occur in $\mathbf{y}_1, \dots, \mathbf{y}_{g(n)}$ and in \mathbf{z}_i that are used to simulate configurations of M^* (see phase (3) below);
- p_t and p_f are propositional variables that do not occur in $\mathbf{q}\mathbf{r}\mathbf{y}$.

(1) Quantification Over Oracles. Next we show how to simulate quantification over oracles. W.l.o.g. we may assume that M^* queries binary strings that are of length $h(n)$ for some polynomial h . Let \mathbf{q} be a sequence of length $h(n)$ and \mathbf{r} a sequence of length $\log(f(n)) + 1$. For $i \in \mathbb{N}$, we let $\text{bin}(i)$ denote a

binary representation of i (with an appropriate number of insignificant zeros). For a string of bits $\mathbf{b} = b_1 \dots b_k$ and a sequence $\mathbf{s} = (s_1, \dots, s_k)$ of proposition symbols, we write $\mathbf{s} = \mathbf{b}$ for $\bigwedge_{i=1}^k s_i = b_i$ where $s_i = 0$ and $s_i = 1$ denote $\neg s_i$ and s_i , respectively. The idea is that, given a team X over \mathbf{qr} , an oracle A_i , and a binary string $\mathbf{a} = a_1 \dots a_{h(n)}$, the membership of \mathbf{a} in A_i is expressed by $X \models \sim \neg(\mathbf{q} = \mathbf{a} \wedge \mathbf{r} = \text{bin}(i))$. Note that the latter indicates that there exists $s \in X$ mapping $\mathbf{q} \mapsto \mathbf{a}$ and $\mathbf{r} \mapsto \text{bin}(i)$. Following this idea we next show how to simulate quantification over oracles A_i . We define φ_i , for $1 \leq i \leq f(n)$, inductively from root to leaves. Depending on whether A_i is existentially or universally quantified, we let

$$\exists : \varphi_i := \mathbf{r} = \text{bin}(i) \vee (\alpha \wedge \varphi_{i+1}), \quad \forall : \varphi_i := \sim \mathbf{r} = \text{bin}(i) \otimes (\sim \alpha \otimes \varphi_{i+1}),$$

where α is defined in $\text{SAT}(\text{PL}[\perp_c, \sim])$ and in $\text{SAT}(\text{PL}[\subseteq, \sim])$ respectively as follows:

$$\begin{aligned} \alpha &:= \max(\mathbf{y}) \wedge \mathbf{y} \perp \mathbf{qr}, \\ \alpha &:= \bigwedge_{i=1}^{|\mathbf{y}|} \mathbf{qr}y_1 \dots y_{i-1}p_f \subseteq \mathbf{qr}y_1 \dots y_i \wedge \bigwedge_{i=1}^{|\mathbf{y}|} \mathbf{qr}y_1 \dots y_{i-1}p_t \subseteq \mathbf{qr}y_1 \dots y_i. \end{aligned}$$

Let us explain the idea behind the definitions of φ_i , first in the case of existential quantification. Assume that we consider a formula φ_i and a team X where

$$X \models \alpha \wedge \varphi_i, \tag{2}$$

and $\{s \in X \mid s(\mathbf{r}) = \text{bin}(i)\}$ is maximal over \mathbf{qy} . Then by (2) we may choose two subsets $Y, Z \subseteq X$, $Y \cup Z = X$, where $Y \models \mathbf{r} = \text{bin}(i)$ and $Z \models \alpha \wedge \varphi_{i+1}$. The idea is that Z must include all assignments $s \in X$ where $s(\mathbf{r}) \neq \text{bin}(i)$, and it may exclude an arbitrary number of assignments $s \in X$ where $s(\mathbf{r}) = \text{bin}(i)$. Hence since $\{s \in X \mid s(\mathbf{r}) = \text{bin}(i)\}$ is maximal over \mathbf{qy} , the set $\{s(\mathbf{q}) \mid s \in Z, s(\mathbf{r}) = \text{bin}(i)\}$ can be chosen to be an arbitrary subset of $\{0, 1\}^{|\mathbf{q}|}$. The only restriction for this choice is that it must be uniform with respect to values of \mathbf{y} , meaning that Z must remain maximal over \mathbf{y} for all \mathbf{qr} . This is ensured by requiring that $Z \models \alpha$.

Universal quantification is simulated analogously. This time we range over all subsets $Y, Z \subseteq X$ where $Y \cup Z = X$. By (2) for all such Y and Z , we have that $Z \models \sim \alpha \otimes \varphi_{i+1}$ if $Y \not\models \sim \mathbf{r} = \text{bin}(i)$. Hence for all subsets Z having only $s \in X$ with $s(\mathbf{r}) = \text{bin}(i)$ removed from it, $Z \models \sim \alpha \otimes \varphi_{i+1}$. This means that such subsets Z satisfy φ_{i+1} whenever they are formed uniformly with respect to values of \mathbf{y} . Analogously to the existential case, we now observe that this corresponds to universal quantification of A_i .

(2) Quantification Over Propositional Variables. Next we show how to simulate the quantifier block $Q'_1 \mathbf{y}_1 \dots Q'_{g(n)} \mathbf{y}_{g(n)} \exists \mathbf{z}$ where \mathbf{z} lists all propositional variables that occur in \mathbf{y} but not in any \mathbf{y}_i (i.e. the remaining variables that occur when simulating M^*). Assume that this quantifier block is of the form $Q_1^* \mathbf{y}_1 \dots Q_l^* \mathbf{y}_l$, and let $\psi_1 := \varphi_{f(n)+1}$. We define ψ_i again top-down inductively. For $1 \leq i \leq l$, depending on whether Q_i^* is \exists or \forall , we let

$$\begin{aligned} \exists: \psi_i &:= \text{dep}(y_i) \vee (\text{dep}(y_i) \wedge \psi_{i+1}), \\ \forall: \psi_i &:= \sim\text{dep}(y_i) \otimes (\sim\text{dep}(y_i) \odot \psi_{i+1}). \end{aligned}$$

Let us explain the idea behind the two definitions of ψ_i , first in the case of existential quantification. Assume that we consider a formula ψ_i and a team X where

$$X \models \alpha \wedge \varphi_i, \tag{3}$$

and X is maximal over $y_i \dots y_l$ for all $\mathbf{q}r y_1 \dots y_{i-1}$. By (3) we may choose two subsets $Y, Z \subseteq X$, $Y \cup Z = X$, where $Y \models \text{dep}(y_i)$ and $Z \models \text{dep}(y_i) \wedge \psi_{i+1}$. There are now two options: either we choose $Z = \{s \in X \mid s(y_i) = 0\}$ or $Z = \{s \in X \mid s(y_i) = 1\}$. Since X is maximal over $y_i \dots y_l$ for all $\mathbf{q}r y_1 \dots y_{i-1}$, we obtain that $Z \upharpoonright \mathbf{q}r = X \upharpoonright \mathbf{q}r$ and Z is maximal over $y_{i+1} \dots y_l$ for all $\mathbf{q}r y_1 \dots y_i$. Hence no information about oracles is lost in this quantifier step.

The case of universal quantification is analogous to the oracle case. Hence we obtain that (3) holds iff both $\{s \in X \mid s(y_i) = 0\}$ and $\{s \in X \mid s(y_i) = 1\}$ satisfy ψ_{i+1} .

(3) Simulation of Computations. Next we define $\psi_{g(n)+1}$ that simulates the polynomial-time deterministic oracle Turing machine M^* . Note that this formula is evaluated over a subteam X such that $X \models \text{dep}(\mathbf{y})$, and $\mathbf{a} \in A_i$ iff $X \models \sim\neg(\mathbf{q} = \mathbf{a} \wedge \mathbf{r} = \text{bin}(i))$. Using this it is now straightforward to construct a propositional formula θ such that $\exists \mathbf{c}(X[\mathbf{b}_i/\mathbf{y}_i][\mathbf{c}/\mathbf{z}] \models \theta)$ if and only if M^* accepts $(x, \mathbf{b}_1, \dots, \mathbf{b}_{g(n)})$ with oracle $(A_1, \dots, A_{f(n)})$. Here $X[\mathbf{a}/\mathbf{x}]$ denotes the team $\{s(\mathbf{a}/\mathbf{x}) : s \in X\}$ where $s(\mathbf{a}/\mathbf{x})$ agrees with s everywhere except that it maps x pointwise to \mathbf{a} . Each configuration of M^* can be encoded with a binary sequence \mathbf{z}_i of length $O(t(n))$ where t is a polynomial bounding the running time of M^* . Then it suffices to define ψ_{l+1} as a conjunction of formulae $\theta_{\text{start}}(\mathbf{z}_0), \theta_{\text{move}}(\mathbf{z}_i, \mathbf{z}_{i+1}), \theta_{\text{final}}(\mathbf{z}_{t(n)})$ describing that \mathbf{z}_0 corresponds to the initial configuration, \mathbf{z}_i determines \mathbf{z}_{i+1} , and $\mathbf{z}_{t(n)}$ is in accepting state. \square

By Proposition 5, and Theorems 5 and 7 we now obtain the following.

Theorem 8. *Satisfiability and validity of $\text{PL}[\perp_c, \sim]$ and $\text{PL}[\subseteq, \sim]$ are complete for $\text{AEXPTIME}(\text{poly})$.*

4 Complexity of Model Checking

In this section we consider the related model checking problems.

Theorem 9. *$\text{MC}(\text{PL}[\perp_c, \sim])$ is complete for NP.*

Proof. The upper bound follows since the model checking problem for modal independence logic is NP-complete [11]. Since dependence atoms can be expressed efficiently by independence atoms (see the proof of Theorem 2), the lower bound follows from the NP-completeness of $\text{MC}(\text{PD})$ (see Table 2).

The following unpublished result was shown by Hella.

Theorem 10. ([9]). $\text{MC}(\text{PL}[\subseteq])$ is in P.

Theorem 11. $\text{MC}(\text{PL}[\sim])$, $\text{MC}(\text{PL}[\perp_c, \cdot])$, and $\text{MC}(\text{PL}[\subseteq, \sim])$ are complete for PSPACE.

Proof. For the upper bounds recall from Theorem 5 that $\text{MC}(\text{PL}[\perp_c, \subseteq, \sim])$ can be decided in APTIME which is exactly PSPACE [2]. For the lower bounds, we reduce to $\text{MC}(\text{PL}[\sim])$ from QBF (quantified Boolean formula problem) which is known to be PSPACE-complete. Let $Q_1x_1 \dots Q_nx_n\theta$ be a quantified boolean formula. Let \mathbf{r} be a sequence of propositional variables such that its length is $\log(n) + 1$, and let $T := \{s_1, \dots, s_n\}$ where $s_i(\mathbf{r})$ writes i in binary. We define inductively top-down a $\varphi \in \text{PL}[\sim]$ such that

$$Q_1x_1 \dots Q_nx_n\theta \text{ is true iff } T \models \varphi. \quad (4)$$

Let $\varphi := \varphi_1$, and for $1 \leq i \leq n$, depending on whether x_i is existentially or universally quantified we let

$$\begin{aligned} \exists: \varphi_i &:= \mathbf{r} = \text{bin}(i) \vee \varphi_{i+1}, \\ \forall: \varphi_i &:= \sim \mathbf{r} = \text{bin}(i) \otimes \varphi_{i+1}. \end{aligned}$$

Finally, we let $\varphi_{n+1} := \theta(\psi_i/x_i)$ where $\psi_i := \sim \neg \mathbf{r} = \text{bin}(i)$, the meaning of which is that s_i exists in the team. Now the above simulation of universal and existential quantification is analogous to that of oracles in the proof of Theorem 7, and hence we may reason that (4) holds. Also T and φ can be constructed in polynomial time, and therefore we obtain the result. \square

5 Conclusion

As it is apparent from the summary of our new contributions depicted in Table 1, a couple open questions still remain:

- Is $\text{VAL}(\text{PL}[\perp_c])$ complete for $\text{coNEXPTIME}^{\text{NP}}$?
- Is $\text{MC}(\text{PL}[\subseteq])$ complete for P?
- Are $\text{SAT}(\text{PL}[\sim])$ and $\text{VAL}(\text{PL}[\sim])$ complete for $\text{AEXPTIME}(\text{poly})$? For this, it would suffice to find a polynomial-time translation of independence or inclusion atom in $\text{PL}[\sim]$.

References

1. Buss, S.: The Boolean formula value problem is in ALOGTIME. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, STOC 1987, pp. 123–131, ACM, New York (1987)
2. Chandra, A.K., Kozen, D.C., Larry, S.J.: Alternation. J. ACM **28**(1), 114–133 (1981)
3. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC 1971, pp. 151–158. ACM, New York (1971)

4. Ebbing, J., Lohmann, P.: Complexity of model checking for modal dependence logic. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012. LNCS, vol. 7147, pp. 226–237. Springer, Heidelberg (2012)
5. Galliani, P.: Inclusion and exclusion dependencies in team semantics: on some logics of imperfect information. *Ann. Pure Appl. Logic* **163**(1), 68–84 (2012)
6. Grädel, E., Väänänen, J.: Dependence and independence. *Stud. Logica* **101**(2), 399–410 (2013)
7. Hannula, M., Kontinen, J.: A finite axiomatization of conditional independence and inclusion dependencies. In: Beierle, C., Meghini, C. (eds.) FoIKS 2014. LNCS, vol. 8367, pp. 211–229. Springer, Heidelberg (2014)
8. Hannula, M., Kontinen, J., Virtema, J., Vollmer, H.: Complexity of propositional independence and inclusion logic. In: *CoRR*, (2015). [abs/1504.06135](https://arxiv.org/abs/1504.06135)
9. Hella, L.: Private communication
10. Hella, L., Kuusisto, A., Meier, A., Vollmer, H.: Modal inclusion logic: Being lax is simpler than being strict. In: Italiano, G.F., et al. (eds.) MFCS 2015, Part I, LNCS, vol. 9234, pp. 281–292. Springer, Heidelberg (2015)
11. Kontinen, J., Müller, J.-S., Schnoor, H., Vollmer, H.: A van Benthem theorem for modal team semantics. In: *CoRR* (2014). [abs/1410.6648](https://arxiv.org/abs/1410.6648)
12. Kontinen, J., Nurmi, V.: Team logic and second-order logic. *Fundam. Inform.* **106**(2–4), 259–272 (2011)
13. Levin, L.A.: Universal search problems. *Probl. Inf. Transm.* **9**(3), 265–266 (1973)
14. Lohmann, P., Vollmer, H.: Complexity results for modal dependence logic. *Stud. Logica* **101**(2), 343–366 (2013)
15. Orponen, P.: Complexity classes of alternating machines with oracles. In: Diaz, J. (ed.) *Automata, Languages and Programming*. LNCS, vol. 154, pp. 573–584. Springer, Heidelberg (1983)
16. Sano, K., Virtema, J.: Axiomatizing propositional dependence logics. In: *CoRR* (2014). [abs/1410.5038](https://arxiv.org/abs/1410.5038)
17. Väänänen, J.: *Dependence Logic*. Cambridge University Press, Cambridge (2007)
18. Virtema, J.: Complexity of validity for propositional dependence logics. In: Peron, A., Piazza, C. (eds.) *Proceedings Fifth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2014, Verona, Italy*. EPTCS, vol. 161, pp. 18–31, 10–12 September 2014
19. Yang, F.: On extensions and variants of dependence logic. Ph.D. thesis, University of Helsinki (2014)
20. Yang, F., Väänänen, J.: Propositional logics of dependence and independence. In: Part I. *CoRR* (2014). [abs/1412.7998](https://arxiv.org/abs/1412.7998)

Modal Inclusion Logic: Being Lax is Simpler than Being Strict

Lauri Hella¹, Antti Kuusisto², Arne Meier³ (✉), and Heribert Vollmer³

¹ School of Information Sciences, University of Tampere,
Kanslerinrinne 1 B, 33014 Tampere, Finland
`lauri.hella@uta.fi`

² Department of Philosophy, Stockholm University, SE-106 91 Stockholm,
Sweden and DTU Compute, Technical University of Denmark,
Richard Petersens Plads 324, 2800 Kgs. Lyngby, Denmark
`antti.j.kuusisto@gmail.com`

³ Institut für Theoretische Informatik, Leibniz Universität Hannover,
Appelstr. 4, 30167 Hannover, Germany
`{meier,vollmer}@thi.uni-hannover.de`

Abstract. We investigate the computational complexity of the satisfiability problem of modal inclusion logic. We distinguish two variants of the problem: one for strict and another one for lax semantics. The complexity of the lax version turns out to be complete for EXPTIME, whereas with strict semantics, the problem becomes NEXPTIME-complete.

1 Introduction

Dependence logic was introduced by Jouko Väänänen [13] in 2007. It is a first-order logic that enables one to explicitly talk about dependencies between variables. It thereby generalizes Henkin quantifiers and also, in a sense, Hintikka's independence-friendly logic. Dependence logic can be used to formalize phenomena from a plethora of scientific disciplines such as database theory, social choice theory, cryptography, quantum physics, and others. It extends first-order logic by specific terms $\text{dep}(x_1, \dots, x_{n-1}, x_n)$ known as dependence atoms, expressing that the value of the variable x_n depends on the values of x_1, \dots, x_{n-1} , i.e., x_n is functionally determined by x_1, \dots, x_{n-1} . As such a dependence does not make sense when talking about single assignments, formulas are evaluated over sets of assignments, called *teams*. The semantics of the atom $\text{dep}(x_1, \dots, x_{n-1}, x_n)$ is defined such that it is true in a team T if in the set of all assignments in T , the value of x_n is functionally determined by the values of x_1, \dots, x_{n-1} .

In addition to dependence atoms, also generalized dependency atoms have been introduced in the literature. Examples include the independence atom (asserting that two sets of variables are informationally independent in a team), the non-emptiness atom (asserting that the team is non-empty), and, most importantly to the present paper, the inclusion atom $\vec{x} \subseteq \vec{y}$ for vectors of variables \vec{x}, \vec{y} , asserting that in a team, the set of tuples assigned to \vec{x} is included

in the set of tuples assigned to \vec{y} . This corresponds to the definition of inclusion dependencies in database theory, which state that all tuples of values taken by the attributes \vec{x} are also taken by the attributes \vec{y} .

Väänänen [14] also introduced dependence atoms into modal logic. There teams are sets of worlds, and a dependence atom $\text{dep}(p_1, \dots, p_{n-1}, p_n)$ holds in a team T if there is a Boolean function that determines the value of p_n from the values of p_1, \dots, p_{n-1} in each world in T . The so obtained modal dependence logic MDL was studied from the point of view of expressivity and complexity in [12]. Following the above mentioned developments in first-order dependence logic, modal dependence logic was also extended by generalized dependency atoms in [7], such as, e.g., independence atoms and inclusion atoms.

In the context of first-order dependence logic and its variants, two alternative kinds of team semantics have been distinguished, *lax* and *strict semantics* [2]. Lax semantics is the standard team semantics, while for strict semantics, some additional uniqueness or strictness properties are required. In the modal context, this mainly concerns the diamond modality \diamond . Usually, i.e., in lax semantics, a formula $\diamond\varphi$ holds in a team T if there is a team S such that every world in T has at least one successor in S and φ holds in S . (Also, the worlds in S are required to have a predecessor in T .) In strict semantics, we require that S contains, for every world in T , a unique successor given by a surjection $f : T \rightarrow S$. (In first-order logic, strict semantics for the existential quantifier is defined similarly.) In both the modal and the first-order context, the operator known as *splitjunction* is also defined differently for lax and strict semantics (see Sect. 2 below).

For many variants of first-order and modal dependence logic, there is no distinction in expressive power between the two semantics. However, the choice of semantics plays a role in independence and inclusion logics, i.e., team semantics over (first-order) logics with the independence and inclusion atoms. For example, in the first-order case, inclusion logic under strict semantics has the same expressive power as dependence logic, i.e., ESO (existential second order logic) [3] and hence NP, while under lax semantics it is equivalent to greatest fixpoint logic and hence can express exactly the polynomial-time decidable properties over finite ordered structures.

The purpose of the present paper is to exhibit a further context in which a quite dramatic difference between the two flavours of team semantics exists. We turn to modal inclusion logic and study the computational complexity of its satisfiability problem. For lax semantics, we show EXPTIME-completeness by proving the upper bound via a translation to a variant of PDL, and the lower bound by a reduction from a succinct encoding of a P-complete problem. Satisfiability under strict semantics is shown NEXPTIME-complete using a translation into two-variable logic with counting (upper bound) and a chain of reductions from a dependence version of QBF-validity (lower bound). The complexity difference also holds for the finite satisfiability problem.

2 Preliminaries

Let Π be a countably infinite set of proposition symbols. The set of formulas of *modal inclusion logic* MInc is defined inductively by the following grammar.

$$\varphi ::= p \mid \neg p \mid (\varphi_1 \wedge \varphi_2) \mid (\varphi_1 \vee \varphi_2) \mid p_1 \cdots p_k \subseteq q_1 \cdots q_k \mid \Box\varphi \mid \Diamond\varphi,$$

where $p, p_1, \dots, p_k, q_1, \dots, q_k \in \Pi$ are proposition symbols and k is any positive integer. The formulas $p_1 \cdots p_k \subseteq q_1 \cdots q_k$ are called *inclusion atoms*. For a set $\Phi \subseteq \Pi$, we let $\text{MInc}(\Phi)$ be the sublanguage where propositions from Φ are used. Observe that formulas are essentially in negation normal form; negations may occur only in front of proposition symbols.

A Kripke model is a structure $M = (W, R, V)$, where $W \neq \emptyset$ is a set (the domain of the model, or the set of worlds/states), $R \subseteq W \times W$ is a binary relation (the accessibility or transition relation), and $V: \Pi \rightarrow \mathcal{P}(W)$ is a *valuation* interpreting the proposition symbols. Here \mathcal{P} denotes the power set operator.

The language of basic unimodal logic is the sublanguage of MInc without formulas $p_1 \cdots p_k \subseteq q_1 \cdots q_k$. We assume that the reader is familiar with standard Kripke semantics of modal logic; we let $M, w \Vdash \varphi$ denote the assertion that the point $w \in W$ of the model M satisfies φ according to standard Kripke semantics. We use the symbol \Vdash in order to refer to satisfaction according to standard Kripke semantics, while the symbol \models will be reserved for *team semantics*, to be defined below, which is the semantics MInc is based on.

Let T be a subset of the domain W of a Kripke model M . The set T is called a *team*. The semantics of the inclusion atoms $p_1 \cdots p_k \subseteq q_1 \cdots q_k$ is defined such that $M, T \models p_1 \cdots p_k \subseteq q_1 \cdots q_k$ iff for each $u \in T$, there exists a point $v \in T$ such that $\bigwedge_{i \in \{1, \dots, k\}} (u \in V(p_i) \Leftrightarrow v \in V(q_i))$. The intuition here is that every

vector of truth values taken by p_1, \dots, p_k , is included in the set of vectors of truth values taken by q_1, \dots, q_k .

Let $M = (W, R, V)$ be a Kripke model and $T \subseteq W$ a team. Define the set of successors of $T \subseteq W$ to be $R(T) := \{s \in W \mid \exists s' \in T : (s', s) \in R\}$. Also define $R\langle T \rangle := \{T' \subseteq W \mid \forall s \in T \exists s' \in T' \text{ s.t. } (s, s') \in R \text{ and } \forall s' \in T' \exists s \in T \text{ s.t. } (s, s') \in R\}$, the set of legal successor teams. The following clauses together with the above clause for inclusion atoms define *lax semantics* for MInc .

$$\begin{aligned} M, T \models^\ell p &\Leftrightarrow w \in V(p) \text{ for all } w \in T. \\ M, T \models^\ell \neg p &\Leftrightarrow w \notin V(p) \text{ holds for all } w \in T. \\ M, T \models^\ell \varphi \wedge \psi &\Leftrightarrow M, T \models^\ell \varphi \text{ and } M, T \models^\ell \psi. \\ M, T \models^\ell \varphi \vee \psi &\Leftrightarrow M, S \models^\ell \varphi \text{ and } M, S' \models^\ell \psi \text{ for some } S, S' \subseteq T \text{ such that} \\ &\quad \text{we have } S \cup S' = T. \\ M, T \models^\ell \Box\varphi &\Leftrightarrow M, R(T) \models^\ell \varphi. \\ M, T \models^\ell \Diamond\varphi &\Leftrightarrow \exists T' \in R\langle T \rangle : M, T' \models^\ell \varphi \end{aligned}$$

The other semantics for MInc , *strict semantics*, differs from the lax semantics only in its treatment of the disjunction \vee and diamond \Diamond . Therefore, all other

clauses in the the definition of \models^s are the same as those for \models^ℓ . The clauses for \vee and \diamond in strict semantics are as follows.

$$\begin{aligned}
 M, T \models^s \varphi \vee \psi &\Leftrightarrow M, S \models^s \varphi \text{ and } M, S' \models^s \psi \text{ for some } S, S' \subseteq T \text{ such that} \\
 &\quad S \cup S' = T \text{ and } S \cap S' = \emptyset. \\
 M, T \models^s \diamond \varphi &\Leftrightarrow M, f(T) \models^s \varphi \text{ for some function } f: T \rightarrow W \text{ such that} \\
 &\quad (u, f(u)) \in R \text{ for all } u \in T. \text{ (Here } f(T) = \{ f(u) \mid u \in T \}.)
 \end{aligned}$$

The difference between lax and strict semantics is as the terms suggest. In strict semantics, the division of a team with the splitjunction \vee is strict; no point is allowed to occur in both parts of the division contrarily to lax semantics. For \diamond , strictness is related to the use of functions when finding a team of successors.

It is well known and easy to show that for a formula φ of modal logic, i.e., a formula of **MInc** *without* inclusion atoms, $M, T \models^\ell \varphi$ iff $\forall w \in T (M, w \Vdash \varphi)$, where \Vdash denotes satisfaction in the standard sense of Kripke semantics. The same equivalence holds for \models^s . This is the so-called *flatness* property.

The satisfiability problem of **MInc** with lax (strict) semantics, is the problem that asks, given a formula φ of **MInc**, whether there exists a nonempty team T and a model such that $M, T \models^\ell \varphi$ ($M, T \models^s \varphi$) holds. Two different problems arise, depending on whether lax or strict semantics is used. The corresponding finite satisfiability problems require that a satisfying model has a finite domain.

3 Computational Complexity

3.1 Upper Bound for Lax Semantics

In this section we show that the satisfiability and finite satisfiability problems of **MInc** with lax semantics are in **EXPTIME**. The result is established by an equivalence preserving translation to *propositional dynamic logic* extended with the global and converse modalities. It is well-known that this logic is complete for **EXPTIME** (see [1, 6, 15]). In fact, we will only need multimodal logic with the global modality and converse modalities for our purposes.

Let Π and \mathcal{R} be countably infinite sets of proposition and binary relation symbols, respectively. We define the following modal language \mathcal{L} via $\varphi ::= p \mid \neg\varphi \mid (\varphi_1 \wedge \varphi_2) \mid \langle R \rangle\varphi \mid \langle R^{-1} \rangle\varphi \mid \langle E \rangle\varphi$. Here $p \in \Pi$, $R \in \mathcal{R}$, and E is a novel symbol. The (classical Kripke-style) semantics of \mathcal{L} is defined with respect to ordinary pointed Kripke models (M, w) for multimodal logic. Let $M = (W, \{R\}_{R \in \mathcal{R}}, V)$ be a Kripke model, where $V: \Pi \rightarrow \mathcal{P}(W)$ is the *valuation* function interpreting proposition symbols. The following clauses define the semantics of \mathcal{L} (notice that we use the turnstile \Vdash instead of \models , which is reserved for team semantics in this paper).

$$\begin{aligned}
 M, w \Vdash p &\Leftrightarrow w \in V(p) \quad \text{and} \quad M, w \Vdash \neg\varphi \Leftrightarrow M, w \not\Vdash \varphi \\
 M, w \Vdash \varphi_1 \wedge \varphi_2 &\Leftrightarrow M, w \Vdash \varphi_1 \text{ and } M, w \Vdash \varphi_2 \\
 M, w \Vdash \langle R \rangle\varphi &\Leftrightarrow M, u \Vdash \varphi \text{ for some } u \text{ such that } wRu \\
 M, w \Vdash \langle R^{-1} \rangle\varphi &\Leftrightarrow M, u \Vdash \varphi \text{ for some } u \text{ such that } uRw \\
 M, w \Vdash \langle E \rangle\varphi &\Leftrightarrow M, u \Vdash \varphi \text{ for some } u \in W
 \end{aligned}$$

We next define a satisfiability preserving translation from modal inclusion logic to \mathcal{L} . We let $[R]$ and $[E]$ denote $\neg\langle R \rangle\neg$ and $\neg\langle E \rangle\neg$, respectively. Before we fix the translation, we define some auxiliary formulas.

Let θ be a formula of MInc . We let $\text{SUB}(\theta)$ denote the set of subformulas of θ ; we distinguish all instances of subformulas, so for example $p \wedge p$ has *three* subformulas (the right and the left instances of p and the conjunction itself). For each formula $\varphi \in \text{SUB}(\theta)$, fix a fresh proposition symbol p_φ that does not occur in θ . We next define, for each $\varphi \in \text{SUB}(\theta)$, a novel auxiliary formula χ_φ .

If $\varphi \in \text{SUB}(\theta)$ is a literal p or $\neg p$, we define $\chi_\varphi := [E](p_\varphi \rightarrow \varphi)$.

Now fix a symbol $R \in \mathcal{R}$, which will ultimately correspond to the diamond used in modal inclusion logic. For the remaining subformulas φ of θ , with the exception of inclusion atoms, the formula χ_φ is defined as follows.

1. $\chi_{\varphi \wedge \psi} := [E]((p_{\varphi \wedge \psi} \leftrightarrow p_\varphi) \wedge (p_{\varphi \wedge \psi} \leftrightarrow p_\psi))$
2. $\chi_{\varphi \vee \psi} := [E](p_{\varphi \vee \psi} \leftrightarrow (p_\varphi \vee p_\psi))$
3. $\chi_{\Box \varphi} := [E]((p_{\Box \varphi} \rightarrow [R]p_\varphi) \wedge (p_\varphi \rightarrow \langle R^{-1} \rangle p_{\Box \varphi}))$
4. $\chi_{\Diamond \varphi} := [E]((p_{\Diamond \varphi} \rightarrow \langle R \rangle p_\varphi) \wedge (p_\varphi \rightarrow \langle R^{-1} \rangle p_{\Diamond \varphi}))$

We then define the formulas χ_α , where $\alpha \in \text{SUB}(\theta)$ is an inclusion atom. We appoint a fresh binary relation R_α for each inclusion atom in θ . Assume α denotes the inclusion atom $p_1 \cdots p_k \subseteq q_1 \cdots q_k$. We define

$$\begin{aligned} \chi_\alpha^+ &:= \bigwedge_{i \in \{1, \dots, k\}} [E]((p_\alpha \wedge p_i) \rightarrow \langle R_\alpha \rangle (p_\alpha \wedge q_i)), \\ \chi_\alpha^- &:= \bigwedge_{i \in \{1, \dots, k\}} [E]((p_\alpha \wedge \neg p_i) \rightarrow \langle R_\alpha \rangle (p_\alpha \wedge \neg q_i)), \\ \chi_\alpha &:= \chi_\alpha^+ \wedge \chi_\alpha^- \wedge \bigwedge_{i \in \{1, \dots, k\}} [E](\langle R_\alpha \rangle q_i \rightarrow [R_\alpha] q_i). \end{aligned}$$

Finally, we define $\varphi_\theta := p_\theta \wedge \bigwedge_{\varphi \in \text{SUB}(\theta)} \chi_\varphi$.

Theorem 1. *The satisfiability and finite satisfiability problems for modal inclusion logic with lax semantics are in EXPTIME.*

Proof. We will show that any formula θ of modal inclusion logic is satisfiable iff its translation φ_θ is. Furthermore, θ is satisfiable over a domain W iff φ_θ is satisfiable over W , whence we also get the desired result for finite satisfiability; \mathcal{L} has the finite model property since it clearly translates to two-variable logic via a simple extension of the *standard translation* (see [1] for the standard translation).

Let $M = (W, R, V)$ be a Kripke model. Let $I(\theta) \subseteq \text{SUB}(\theta)$ be the set of inclusion atoms in θ . Assume that $M, X \models^\ell \theta$, where X is a nonempty team. We next define a multimodal Kripke model $N := (W, R, \{R_\alpha\}_{\alpha \in I(\theta)}, V \cup U)$, where $U: \{p_\varphi \mid \varphi \in \text{SUB}(\theta)\} \rightarrow \mathcal{P}(W)$ extends the valuation function V .

Define $U(p_\theta) = X$. Thus we have $M, U(p_\theta) \models^\ell \theta$. Working from the root towards the leaves of the parse tree of θ , we next interpret the remaining predicates p_φ inductively such that the condition $M, U(p_\varphi) \models^\ell \varphi$ is maintained.

Assume $U(p_{\psi \wedge \psi'})$ has been defined. We define $U(p_\psi) = U(p_{\psi'}) = U(p_{\psi \wedge \psi'})$. As $M, U(p_{\psi \wedge \psi'}) \models^\ell \psi \wedge \psi'$, we have $M, U(p_\psi) \models^\ell \psi$ and $M, U(p_{\psi'}) \models^\ell \psi'$.

Assume then that $U(p_{\psi \vee \psi'})$ has been defined. Thus there exist sets S and S' such that $M, S \models^\ell \psi$ and $M, S' \models^\ell \psi'$, and furthermore, $S \cup S' = U(p_{\psi \vee \psi'})$. We define $U(p_\psi) = S$ and $U(p_{\psi'}) = S'$. Consider then the case where $U(p_{\diamond\varphi})$ has been defined. Call $T := U(p_{\diamond\varphi})$. As $M, T \models^\ell \diamond\varphi$, there exists a set $T' \subseteq W$ such that each point in T has an R -successor in T' , and each point in T' has an R -predecessor in T , and furthermore, $M, T' \models^\ell \varphi$. We set $U(p_\varphi) := T'$. Finally, in the case for $p_{\square\varphi}$, the set $U(p_\varphi)$ is defined to be the set of points that have an R -predecessor in $U(p_{\square\varphi})$.

We have now fixed an interpretation for each of the predicates p_φ . The relations R_α , where α is an inclusion atom, remain to be interpreted. Let $p_1 \cdots p_k \subseteq q_1 \cdots q_k$ be an inclusion atom in θ , and denote this atom by α . Call $T := U(p_\alpha)$. Let $u \in T$. Since $M, T \models^\ell \alpha$, there exists a point $v \in T$ such that for each $i \in \{1, \dots, k\}$, $u \in V(p_i)$ iff $v \in V(q_i)$. Define the pair (u, v) to be in R_α . In this fashion, consider each point u in T and find exactly one corresponding point v for u , and put the pair (u, v) into R_α . This fixes the interpretation of R_α .

Let $w \in X = U(p_\theta)$. Recalling how the sets $U(p_\varphi)$ were defined, it is now routine to check that $N, w \Vdash \varphi_\theta$.

We then consider the converse implication of the current theorem. Thus we assume that $N, w \Vdash \varphi_\theta$, where N is some multimodal Kripke model in the signature of φ_θ and w a point in the domain of N . We let W denote the domain and V the valuation function of N .

For each $\varphi \in SUB(\theta)$, define the team $X_\varphi := V(p_\varphi)$. We will show by induction on the structure of θ that for each $\varphi \in SUB(\theta)$, we have $N, X_\varphi \models^\ell \varphi$. Once this is done, it is clear that $M, X_\theta \models^\ell \theta$, where M is the restriction of N to the signature of θ , and we have $X_\theta \neq \emptyset$.

Now recall the definition of the formulas χ_φ , where $\varphi \in SUB(\theta)$. Let $p \in SUB(\theta)$. It is clear that $N, X_p \models^\ell p$, since $N, w \Vdash \chi_p$. Similarly, we infer that $N, X_{\neg q} \models^\ell \neg q$ for $\neg q \in SUB(\theta)$.

Consider then a subformula $p_1 \cdots p_k \subseteq q_1 \cdots q_k$ of φ . Denote this inclusion atom by α . Consider a point $u \in X_\alpha$. If u satisfies p_i for some $i \in \{1, \dots, k\}$, then we infer that since $N, w \Vdash \chi_\alpha^+$, there exists a point $v_i \in X_\alpha$ that satisfies q_i . Similarly, if u satisfies $\neg p_j$, we infer that since $N, w \Vdash \chi_\alpha^-$, there exists a point $v_j \in X_\alpha$ that satisfies $\neg q_j$. To conclude that $N, X_\alpha \models^\ell \alpha$, it suffices to show that all such points v_i and v_j can be chosen such that $v_i = v_j$ for all $i, j \in \{1, \dots, k\}$. This follows due to the third conjunct of χ_α .

Having established the basis of the induction, the rest of the argument is straightforward. We consider explicitly only the case where the subformula under consideration is $\diamond\varphi$. Here we simply need to argue that for each $u \in X_{\diamond\varphi}$, there exists a point $v \in X_\varphi$ such that uRv , and for each $u' \in X_\varphi$, there exists a point $v' \in X_{\diamond\varphi}$ such that $v'Ru'$. This follows directly, since $N, w \Vdash \chi_{\diamond\varphi}$. \square

3.2 Lower Bound for Lax Semantics

In this section we prove that the satisfiability problem of MInc with lax semantics, MInc-lax-SAT , is hard for EXPTIME . We do this by reducing the succinct version

of the following P-hard problem to it which is closely related to the problem PATH SYSTEMS [4, p. 171].

Definition 1. Let PER be the following problem: An instance of PER is a structure $\mathfrak{A} = (A, S)$ with $A = \{1, \dots, n\}$ and $S \subseteq A^3$. A subset P of A is S -persistent if it satisfies the condition (*) if $i \in P$, then there are $j, k \in P$ such that $(i, j, k) \in S$. \mathfrak{A} is a positive instance if $n \in P$ for some S -persistent set $P \subseteq A$.

It is well known that structures (A, S) as above can be represented in a succinct form by using Boolean circuits. Namely if C is Boolean circuit with $3 \cdot l$ input gates then it defines a structure $\mathfrak{A}_C = (A_C, S_C)$ given below. We use here the notation $\#(a_1, \dots, a_l)$ for the natural number i , whose binary representation is (a_1, \dots, a_l) . Let $A_C = \{1, \dots, 2^l\}$, and for all $i, j, k \in A$, let $(i, j, k) \in S_C$ if and only if C accepts the input tuple $(a_1, \dots, a_l, b_1, \dots, b_l, c_1, \dots, c_l) \in \{0, 1\}^{3l}$, where $i = \#(a_1, \dots, a_l)$, $j = \#(b_1, \dots, b_l)$ and $k = \#(c_1, \dots, c_l)$. We say that C is a succinct representation of \mathfrak{A}_C .

Definition 2. The succinct version of PER, S-PER, is the following problem: An instance of S-PER is a circuit C with $3l$ input gates. C is a positive instance, if \mathfrak{A}_C is a positive instance of PER.

Proposition 1. S-PER is EXPTIME-hard with respect to PSPACE reductions.

Proof. (Idea.) The succinct version of the CIRCUIT VALUE problem is polynomial space reducible to S-PER. Since succinct CIRCUIT VALUE is known to be EXPTIME-complete (see [9, Sect.20]), the claim follows. For the details of the proof, see the technical report [5]. □

We will next show that S-PER is polynomial time reducible to the satisfiability problem of MInc with lax semantics, and hence the latter is also EXPTIME-hard. In the proof we use the following notation: If T is a team and p_1, \dots, p_n are proposition symbols, then $T(p_1, \dots, p_n)$ is the set of all tuples $(a_1, \dots, a_n) \in \{0, 1\}^n$ such that for some $w \in T$, $a_t = 1 \iff w \in V(p_t)$ for $t \in \{1, \dots, n\}$. Note that the semantics of inclusion atoms can now be expressed as

$$M, T \models p_1 \cdots p_n \subseteq q_1 \cdots q_n \iff T(p_1, \dots, p_n) \subseteq T(q_1, \dots, q_n).$$

Theorem 2. The satisfiability and finite satisfiability problems for MInc with lax semantics are hard for EXPTIME with respect to PSPACE reductions.

Proof. Let C be a Boolean circuit with $3l$ input gates. Let g_1, \dots, g_m be the gates of C , where g_1, \dots, g_{3l} are the input gates and g_m is the output gate. We fix a distinct Boolean variable p_i for each gate g_i . Let Φ be the set $\{p_1, \dots, p_m\}$ of proposition symbols. We define for each $i \in \{3l + 1, \dots, m\}$ a formula $\theta_i \in \text{MInc}(\Phi)$ that describes the correct operation of the gate g_i :

$$\theta_i = \begin{cases} p_i \leftrightarrow \neg p_j & \text{if } g_i \text{ is a NOT gate with input } g_j \\ p_i \leftrightarrow (p_j \wedge p_k) & \text{if } g_i \text{ is an AND gate with inputs } g_j \text{ and } g_k \\ p_i \leftrightarrow (p_j \vee p_k) & \text{if } g_i \text{ is an OR gate with inputs } g_j \text{ and } g_k \end{cases}$$

Let ψ_C be the formula $(\bigwedge_{3l+1 \leq i \leq m} \theta_i) \wedge p_m$. Thus, ψ_C essentially says that the truth values of p_i , $1 \leq i \leq m$, match an accepting computation of C .

Now we can define a formula φ_C of $\text{MInc}(\Phi)$ which is satisfiable if and only if C is a positive instance of S-PER. For the sake of readability, we denote here the variables corresponding to the input gates g_{l+1}, \dots, g_{2l} by q_1, \dots, q_l . Similarly, we denote the variables p_{2l+1}, \dots, p_{3l} by r_1, \dots, r_l .

$$\varphi_C := \psi_C \wedge q_1 \cdots q_l \subseteq p_1 \cdots p_l \wedge r_1 \cdots r_l \subseteq p_1 \cdots p_l \wedge p_m \cdots p_m \subseteq p_1 \cdots p_l.$$

Note that φ_C can clearly be constructed from the circuit C in polynomial time.

Assume first that φ_C is satisfiable. Thus there is a Kripke model $M = (W, R, V)$ and a nonempty team T of M such that $M, T \models^\ell \varphi_C$. Consider the model $\mathfrak{A}_C = (A_C, S_C)$ that corresponds to the circuit C . We define a subset P of A_C as follows: $P := \{\#(a_1, \dots, a_l) \mid (a_1, \dots, a_l) \in T(p_1, \dots, p_l)\}$.

Observe first that since $M, T \models^\ell p_m$ and $M, T \models^\ell p_m \cdots p_m \subseteq p_1 \cdots p_l$, $(1, \dots, 1) \in T(p_1, \dots, p_l)$ and hence $2^l = \#(1, \dots, 1) \in P$. Thus, it suffices to show that P is S_C -persistent. To prove this, assume that $i = \#(a_1, \dots, a_l) \in P$. Then there is a state $w \in T$ such that $w \in V(p_t) \iff a_t = 1$ for $1 \leq t \leq l$.

Define now $b_t, c_t \in \{0, 1\}$, $1 \leq t \leq l$, by the condition

$$b_t = 1 \iff w \in V(q_t) \quad \text{and} \quad c_t = 1 \iff w \in V(r_t).$$

As $M, T \models^\ell \psi_C$, it follows from flatness that $M, w \models \psi_C$. By the definition of ψ_C , this means that the circuit C accepts the input tuple $(a_1, \dots, a_l, b_1, \dots, b_l, c_1, \dots, c_l)$. Thus, $(i, j, k) \in S_C$, where $j = \#(b_1, \dots, b_l)$ and $k = \#(c_1, \dots, c_l)$.

We still need to show that $j, k \in P$. To see this, note that since $M, T \models^\ell q_1 \cdots q_l \subseteq p_1 \cdots p_l$, there exists $w' \in T$ such that

$$w' \in V(p_t) \iff w \in V(q_t) \iff b_t = 1 \quad \text{for } 1 \leq t \leq l.$$

Thus, $(b_1, \dots, b_l) \in T(p_1, \dots, p_n)$, whence $j \in P$. Similarly we see that $k \in P$.

To prove the other implication, assume that C is a positive instance of the problem S-PER. Then there is an S_C -persistent set $P \subseteq A_C$ such that $2^l \in P$. We let $M = (W, R, V)$ be the Kripke model and T the team of M such that

- $T = W$ is the set of all tuples $(a_1, \dots, a_m) \in \{0, 1\}^m$ that correspond to an accepting computation of C and for which $\#(a_1, \dots, a_l), \#(a_{l+1}, \dots, a_{2l}), \#(a_{2l+1}, \dots, a_{3l}) \in P$,
- $R = \emptyset$, and $V(p_t) = \{(a_1, \dots, a_m) \in W \mid a_t = 1\}$ for $1 \leq t \leq m$.

We will now show that $M, T \models^\ell \varphi_C$, whence φ_C is satisfiable. Note first that $M, T \models^\ell \psi_C$, since by the definition of T and V , for any $w \in T$, the truth values of p_i in w correspond to an accepting computation of C .

To prove $M, T \models^\ell q_1 \cdots q_l \subseteq p_1 \cdots p_l$, assume that $(b_1, \dots, b_l) \in T(q_1, \dots, q_l)$. Then $i := \#(b_1, \dots, b_l) \in P$, and since P is S_C -persistent, there are $j, k \in P$ such that $(i, j, k) \in S_C$. Thus, there is a tuple $(a_1, \dots, a_m) \in \{0, 1\}^m$ corresponding to an accepting computation of C such that $(a_1, \dots, a_l) = (b_1, \dots, b_l)$, $j = \#(a_{l+1}, \dots, a_{2l})$ and $k = \#(a_{2l+1}, \dots, a_{3l})$. This means that (a_1, \dots, a_m) is in T ,

and hence $(b_1, \dots, b_l) \in T(p_1, \dots, p_l)$. The claim $M, T \models^\ell r_1 \cdots r_l \subseteq p_1 \cdots p_l$ is proved in the same way.

Note that since $M, T \models p_m$, we have $T(p_m, \dots, p_m) = \{(1, \dots, 1)\}$. Furthermore, since $2^l = \sharp(1, \dots, 1) \in P$ and P is S_C -persistent, there is an element $(a_1, \dots, a_m) \in T$ such that $(a_1, \dots, a_l) = (1, \dots, 1)$. Thus, we see that $(1, \dots, 1) \in T(p_1, \dots, p_l)$, and consequently $M, T \models^\ell p_m \cdots p_m \subseteq p_1 \cdots p_l$. \square

Corollary 1. *The satisfiability and finite satisfiability problems of modal inclusion logic with lax semantics are EXPTIME-complete with respect to PSPACE reductions.*

Note that the formula φ_C used in the proof of Theorem 2 is in *propositional inclusion logic*, i.e., it does not contain any modal operators. Thus, our proof shows that the satisfiability problem of propositional inclusion logic is already EXPTIME-hard. Naturally, this problem is also in EXPTIME, since propositional inclusion logic is a fragment of MInc.

Corollary 2. *The satisfiability and finite satisfiability problems of propositional inclusion logic with lax semantics are EXPTIME-complete with respect to PSPACE reductions.*

3.3 Upper Bound for Strict Semantics

In this section we show that the satisfiability and finite satisfiability problems for MInc with strict semantics are in NEXPTIME. The proof is a simple adaptation of the upper bound argument for lax semantics, but uses *two-variable logic with counting*, FOC², which has NEXPTIME-complete satisfiability and finite satisfiability problems [11] (but no finite model property).

Let θ be a formula of MInc. The equisatisfiable translation of θ is obtained from the formula φ_θ , which we defined when considering lax semantics. It is clear that φ_θ translates via a simple extension of the *standard translation* into FOC²; see [1] for the standard translation of modal logic. Let $t(\varphi_\theta)$ denote the FOC²-formula obtained by using the (extension of the) standard translation. For each $\varphi \in SUB(\varphi_\theta)$, let $t(\chi_\varphi)$ denote the translation of the subformula χ_φ of φ_θ ; see the argument for lax semantics for the definition of the formulas χ_φ . The only thing we now need to do is to modify the formulas $t(\chi_{\diamond\varphi})$ and $t(\chi_{\varphi\vee\psi})$.

In the case of $t(\chi_{\varphi\vee\psi})$, we simply add a conjunct stating that the unary predicates p_φ and p_ψ are interpreted as disjoint sets: $\neg\exists x(p_\varphi(x) \wedge p_\psi(x))$.

To modify the formulas $t(\chi_{\diamond\varphi})$, we appoint a novel binary relation $R_{\diamond\varphi}$ for each formula $\diamond\varphi \in SUB(\theta)$. We then define the formula β which states that $R_{\diamond\varphi}$ is a function from the interpretation of $p_{\diamond\varphi}$ onto the interpretation of p_φ .

$$\begin{aligned} \beta := \forall x(p_{\diamond\varphi}(x) \rightarrow \exists^{=1}y(R_{\diamond\varphi}xy \wedge p_\varphi(y)) \wedge \forall x\forall y(R_{\diamond\varphi}xy \rightarrow (p_{\diamond\varphi}(x) \wedge p_\varphi(y))) \\ \wedge \forall y(p_\varphi(y) \rightarrow \exists x(p_{\diamond\varphi}(x) \wedge R_{\diamond\varphi}xy)). \end{aligned}$$

Define $\beta' := \forall x\forall y(R_{\diamond\varphi}xy \rightarrow Rxy)$, where R is the accessibility relation of modal inclusion logic. The conjunction $\beta \wedge \beta'$ is the desired modification of $t(\chi_{\diamond\varphi})$.

The modification of $t(\varphi_\theta)$, using the modified versions of $t(\chi_{\varphi \vee \psi})$ and $t(\chi_{\diamond \varphi})$, is the desired FOC²-formula equisatisfiable with θ . The proof of the following theorem is practically identical to the corresponding argument for lax semantics.

Theorem 3. *The satisfiability and finite satisfiability problems for MInc with strict semantics are in NEXPTIME.*

3.4 Lower Bound for Strict Semantics

Theorem 4. *The satisfiability and finite satisfiability problems for MInc with strict semantics are NEXPTIME-hard.*

Proof. We will provide a chain of reductions from *Dependence-QBF-Validity* (in short DQBF-VAL) to *Inclusion-QBF-Validity* (in short IncQBF-VAL), and finally to satisfiability of MInc with strict semantics.

Peterson et al. [10] introduced a so-to-speak dependence version of QBF by extending the usual QBF syntax to allow stating on which universally quantified propositions an existentially quantified proposition solely depends. Instances of the problem are of the form $(\forall p_1)(\exists q_1 \setminus P_1) \cdots (\forall p_k)(\exists q_k \setminus P_k) \varphi$ (\star), where each set P_i contains a subset of the propositions $\{p_1, \dots, p_i\}$ quantified universally superordinate to q_i , and φ is a propositional logic formula in the variables $\{p_1, \dots, p_k\} \cup \{q_1, \dots, q_k\}$. The set P_i indicates that the choice for the value of q_i is given by a Boolean function that takes as inputs only the values of the variables in P_i (see [10] for the full details).

By well-known standard arguments in the field of team semantics, it is easy to show that the formula of Eqn. (\star) can be written in the alternative form (where \bar{p}_i lists the variables in P_i)

$$(\forall p_1)(\exists q_1) \cdots (\forall p_k)(\exists q_k) (\varphi \wedge \bigwedge_{i \in \{1, \dots, k\}} \text{dep}(\bar{p}_i, q_i)), \tag{1}$$

with the following semantics (where M is a Kripke model and T is a team).

- $M, T \models \forall p \psi$ iff $M', T^p \models \psi$, where T^p is obtained from T by simultaneously replacing each $w \in T$ by two new worlds u, v that agree with w on all propositions other than p , and the points u, v disagree with each other on p . M' is obtained from M by modifying the domain W of M to the new domain $W' = T^p \cup (W \setminus T)$, and modifying the valuation of M to a new one that agrees with the specification of T^p ; outside T^p the new valuation agrees with the old one. The accessibility relation does not play a role here.
- $M, T \models \exists p \psi$ iff $M', T_p \models \psi$, where T_p is obtained from T by simultaneously replacing each $w \in T$ by a new world u that agrees with w on propositions other than p , and may or may not agree with w on p . Similarly to the case above, M' is obtained from M by modifying the domain W of M to the new domain $W' = T_p \cup (W \setminus T)$, and modifying the valuation of M to a new one that agrees with the specification of T_p ; outside T_p the new valuation agrees with the old one. The accessibility relation does not play a role here.

- The connectives \vee and \wedge are interpreted exactly as in the case of modal inclusion logic using strict semantics. Literals p , $\neg p$ are also interpreted as in modal inclusion logic.
- $M, T \models \text{dep}(p_1, \dots, p_k, q)$ if each pair of worlds in T that agree on the truth values of each of the propositions p_1, \dots, p_k , also agree on the value of q .

Our formulation of the DQBF-VAL problem of Peterson et al. [10], with alternative inputs such as those in Eq. 1, is equivalent to the original problem. Peterson et al. showed that their problem lifts the computational complexity from PSPACE-completeness (for the standard quantified Boolean formula validity) to in fact NEXPTIME-completeness.

Inclusion-QBF (IncQBF) is a language obtained from our formulation of the Dependence-QBF (DQBF). It translates the expressions $\text{dep}(p_1, \dots, p_k, q)$ to inclusion atoms, in the way we next describe. Inspired by Galliani et al. [3], we observe that inclusion atoms can simulate formulas $\text{dep}(p_1, \dots, p_k, q)$, as the following example demonstrates: $\forall p \forall q \exists r (\text{dep}(q, r) \wedge \varphi)$ is equivalent to $\forall p \forall q \exists r (\forall s (sqr \subseteq pqr) \wedge \varphi)$, where φ is a formula of propositional logic. This can be generalized to work for expressions with conjunctions of atoms $\text{dep}(p_1, \dots, p_k, q)$ for arbitrary k .

Now, for the last step, we need to explain how IncQBF-VAL finally reduces to MInc-strict-SAT. This is just a slight modification of the standard proof of Ladner showing PSPACE-hardness of plain modal logic via a reduction from QBF validity [8]. The idea is to enforce a complete assignment tree. Further, one uses clause propositions which are true if the corresponding literal holds. Let us denote the formula which enforces the described substructure by φ_{struc} (for details, see [8]). The final formula is obtained from an IncQBF-VAL instance $\exists r_1 \forall r_2 \dots \exists r_n (\varphi \wedge \chi)$ where φ is the conjunctive normal form formula and χ is the conjunction of the inclusion atoms (stemming from the translation above); the final formula is then a formula of type $\varphi_{\text{struc}} \wedge \diamond \square \dots \Delta (\varphi \wedge \chi)$, where $\Delta = \square$ if $\exists = \forall$ and $\Delta = \diamond$ if $\exists = \exists$. Let us denote this translation by the function f which can be computed in polynomial time. Then it is easy to verify that $\varphi \in \text{IncQBF-VAL}$ iff $f(\varphi) \in \text{MInc-strict-SAT}$. It is straightforward to observe that this covers also the case for finite satisfiability. \square

Corollary 3. *The satisfiability and finite satisfiability problems of modal inclusion logic with strict semantics are NEXPTIME-complete.*

4 Conclusion

We have compared the strict and lax variants of team semantics from the perspective of satisfiability problems for modal inclusion logic MInc. Interestingly, the problems differ in complexity. Strict semantics leads to NEXPTIME-completeness, while lax semantics gives completeness for EXPTIME. For the journal version we plan to include a stronger polynomial-time reduction result for the EXPTIME lower bound of MInc-lax-SAT. In the future it will be interesting to study model checking problems for MInc under strict and lax semantics.

Also, the complexity of validity problems for MInc and, related to this, proof-theoretic properties of the logic remain to be investigated.

Acknowledgements. The authors thank the anonymous referees for their comments. The third author is supported by DFG grant ME 4279/1-1. The second author acknowledges support from Jenny and Antti Wihuri Foundation.

References

1. Blackburn, P., de Rijke, M., Venema, Y.: *Modal Logic*. Cambridge University Press, Cambridge (2001)
2. Galliani, P.: Inclusion and exclusion dependencies in team semantics - on some logics of imperfect information. *Ann. Pure Appl. Logic* **163**(1), 68–84 (2012)
3. Galliani, P., Hannula, M., Kontinen, J.: Hierarchies in independence logic. In: Ronchi Della Rocca, S. (ed.) *Proceedings of Computer Science Logic 2013. Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 23, pp. 263–280 (2013)
4. Greenlaw, R., Hoover, H.J., Ruzzo, W.L.: *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, Oxford (1995)
5. Hella, L., Kuusisto, A., Meier, A., Vollmer, H.: Modal inclusion logic: Being lax is simpler than being strict. *arXiv*, 1504.06409 (2015)
6. Hemaspaandra, E.: The price of universality. *Notre Dame J. Formal Logic* **37**(2), 174–203 (1996)
7. Kontinen, J., Müller, J.-S., Schnoor, H., Vollmer, H.: Modal independence logic. In: Goré, R., Kooi, B.P., Kurucz, A. (eds.) *Proceedings of Advances in Modal Logic*, vol. 10, pp. 353–372. College Publications, London (2014)
8. Ladner, R.: The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput.* **6**(3), 467–480 (1977)
9. Papadimitriou, C.H.: *Computational Complexity*. Addison-Wesley, Reading (1994)
10. Peterson, G., Azhar, S., Reif, J.H.: Lower bounds for multiplayer noncooperative games of incomplete information. *Comput. Math. Appl.* **41**, 957–992 (2001)
11. Pratt-Hartmann, I.: Complexity of the two-variable fragment with counting quantifiers. *JoLLI* **14**(3), 369–395 (2005)
12. Sevenster, M.: Model-theoretic and computational properties of modal dependence logic. *J. Logic Comput.* **19**(6), 1157–1173 (2009)
13. Väänänen, J.: *Dependence Logic*. Cambridge University Press, Cambridge (2007)
14. Väänänen, J.: Modal dependence logic. In: Apt, K., van Rooij, R. (eds.) *New Perspectives on Games and Interaction*, pp. 237–254. Amsterdam University Press, Amsterdam (2008)
15. van Eijck, J.: Dynamic epistemic logics. In: Baltag, A., Smets, S. (eds.) *Johan van Benthem on Logic and Information Dynamics*, pp. 175–202. Springer, Heidelberg (2014)

Differential Bisimulation for a Markovian Process Algebra

Giulio Iacobelli¹, Mirco Tribastone^{2(✉)}, and Andrea Vandin²

¹ Computing and Systems Engineering, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil

² IMT Institute for Advanced Studies Lucca, Lucca, Italy
`mirco.tribastone@imtlucca.it`

Abstract. Formal languages with semantics based on ordinary differential equations (ODEs) have emerged as a useful tool to reason about large-scale distributed systems. We present *differential bisimulation*, a behavioral equivalence developed as the ODE counterpart of bisimulations for languages with probabilistic or stochastic semantics. We study it in the context of a Markovian process algebra. Similarly to Markovian bisimulations yielding an aggregated Markov process in the sense of the theory of lumpability, differential bisimulation yields a partition of the ODEs underlying a process algebra term, whereby the sum of the ODE solutions of the same partition block is equal to the solution of a single (lumped) ODE. Differential bisimulation is defined in terms of two symmetries that can be verified only using syntactic checks. This enables the adaptation to a continuous-state semantics of proof techniques and algorithms for finite, discrete-state, labeled transition systems. For instance, we readily obtain a result of compositionality, and provide an efficient partition-refinement algorithm to compute the coarsest ODE aggregation of a model according to differential bisimulation.

1 Introduction

There has been increasing attention to models of computation based on ordinary differential equations (ODEs). This has been mainly prompted by a line of research which interprets an ODE as the deterministic (called *fluid* or *mean-field*) approximation [16, 17] of a continuous time Markov chain (CTMC) underlying languages with Markovian semantics [6, 11, 24]. The ODE semantics provides the behavior of a (concurrent) program as a continuous trajectory representing the *concentration* of processes over time.

In this paper we consider the following problem: *How to compare programs with ODE semantics?* Our main contribution is to lift the notion of bisimulation to languages with ODE semantics. To put it in context, let us draw a parallel with

This work was partially supported by the EU project QUANTICOL, 600708. The postdoctoral fellowship of G.I. is supported by CAPES. Part of this research has been carried out while the three authors were at University of Southampton, UK.

established results of aggregation of CTMCs obtained from a Markovian semantics of a high-level language such as process algebra (e.g., [2, 4, 14]). This involved finding behavioural relations that induce a partition of the CTMC states which satisfies the property of *ordinary lumpability* [3]: a smaller CTMC can be constructed where each state (a *macro-state*) is the representative of the states in a block; the probability of being in a macro-state is equal to the sum of those of being in the block's states. Here we proceed analogously. We introduce *differential bisimulation* (DB), an equivalence relation that captures symmetries in the ODE semantics according to the well-known theory of ODE *lumpability* [23]: the solution to each ODE representing an equivalence class is equal at all time points to the sum of the solutions of the ODEs of the states in that equivalence class.

We study DB for Fluid Extended Process Algebra (FEPA) [25], a fragment of PEPA [14] with ODE semantics, extended to also capture the product-based synchronisation mechanism of [4, 12]. A FEPA model is a composition of *fluid atoms*, each representing a population of identical copies in parallel of the same sequential process, describing its evolution over its set of *local states*. The interaction between fluid atoms occurs via shared channels. A FEPA model encodes a family of systems, parametric in the population sizes of each fluid atom. Under appropriate scaling conditions each member is represented by the same ODEs, one for each local state of each fluid atom, giving the evolution of the number of sequential processes exhibiting that local state.

Differential bisimulation is an equivalence relation over local states of a process. This is in contrast to Markovian bisimulations, which are defined over states of a CTMC. However, DB can be seen as a natural generalization. Indeed it consists of two conditions, the first of which is essentially a Larsen-Skou style bisimulation (cf. [18]) over local states. When a process consists of a fluid atom with one replica (i.e., a single sequential process), the ODE and the CTMC semantics coincide, and DB collapses onto strong equivalence, PEPA's Markovian bisimulation. In the CTMC case such a condition suffices to imply lumpability, informally because the CTMC transition diagram of a process term with an arbitrary synchronization tree structure is isomorphic to the transition system of a single sequential process (by mapping each CTMC state to a named choice term). In the ODE semantics, instead, the synchronization structure is encoded in the function governing the ODE evolution. This is taken into account with the second condition of DB: we introduce the novel concept of *structural interface*, an equivalence relation for local states with same capability to interact with the environment. Both conditions can be checked statically, i.e., syntactically over the process term. Due to the relation with Markovian bisimulation, it is possible to adapt partition-refinement algorithms available for discrete-state labeled transition systems (e.g. [1, 13, 20]), offering an efficient way to compute the coarsest ODE aggregation of a model up to DB.

2 Preliminaries: FEPA

The grammar of FEPA has two levels. The first level specifies a *fluid atom*, i.e. a sequential process evolving over a discrete state space. Let \mathcal{A} denote the set

of actions and \mathcal{K} the set of constants. Each $P \in \mathcal{K}$ is a *sequential component*, defined as $P \stackrel{def}{=} \sum_{i \in I_P} (\alpha_i, r_i).P_i$, where I_P is an index set, $\alpha_i \in \mathcal{A}$, $r_i \in \mathbb{R}_{\geq 0}$ is a rate, and $P_i \in \mathcal{K}$. The multi-set of outgoing transitions from P , denoted by $out(P)$, is defined as the one containing a transition $P \xrightarrow{(\alpha_i, r_i)} P_i$ for each occurrence of $(\alpha_i, r_i).P_i$ in the definition of P . We now define the second level of the grammar. The parallel operator is parameterised by a binary *synchronisation function*, denoted by $\mathcal{H}(\cdot, \cdot)$. As discussed, we support two such functions, $\mathcal{H} = \min$ and $\mathcal{H} = \cdot$ (product). According to the chosen interpretation, fluid atoms may correspond to, e.g., jobs and servers in a computing system, or to molecular species in a chemical reaction network.

Definition 1 (FEPA Model). *A FEPA model \mathcal{M} is generated by*

$$\mathcal{M} ::= P : \mathcal{M} \parallel_L^{\mathcal{H}} \mathcal{M}, \quad \text{with } L \subseteq \mathcal{A} \text{ and } P \in \mathcal{K}$$

Let $\mathcal{G}(\mathcal{M})$ be the set of *fluid atoms* of a FEPA model \mathcal{M} , recursively defined as $\mathcal{G}(P) = \{P\}$, and $\mathcal{G}(\mathcal{M}_1 \parallel_L^{\mathcal{H}} \mathcal{M}_2) = \mathcal{G}(\mathcal{M}_1) \cup \mathcal{G}(\mathcal{M}_2)$. For $P \in \mathcal{G}(\mathcal{M})$, the *local states* of P , denoted $\mathcal{B}(P)$, are the smallest set such that $P \in \mathcal{B}(P)$ and if $P' \in \mathcal{B}(P)$ and $P' \xrightarrow{(\alpha, r)} P'' \in out(P')$, then $P'' \in \mathcal{B}(P)$. We use $\mathcal{B}(\mathcal{M})$ for $\bigcup_{P \in \mathcal{G}(\mathcal{M})} \mathcal{B}(P)$. For any two $P, Q \in \mathcal{G}(\mathcal{M})$, we assume $\mathcal{B}(P) \cap \mathcal{B}(Q) = \emptyset$. This is without loss of generality (e.g., by renaming with fresh variables). For $P \in \mathcal{B}(\mathcal{M})$ we use $\mathcal{A}(P)$ for the set of actions labeling transitions from P . The compositional operator $\parallel_L^{\mathcal{H}}$, parametrized by an action set and by the function \mathcal{H} , specifies the type of synchronisation and the channels used for interaction. Notably, different instantiations of \mathcal{H} can appear in a FEPA model.

Example 1. Let $\mathcal{M}_F \triangleq P_1 \parallel_{\{\alpha\}}^{\mathcal{H}} Q_1$, with P_1, Q_1 defined as

$$\begin{aligned} P_1 &\stackrel{def}{=} (\beta, r).P_2 + (\beta, r).P_3, & P_2 &\stackrel{def}{=} (\alpha, s).P_1, & P_3 &\stackrel{def}{=} (\alpha, s).P_1 \\ Q_1 &\stackrel{def}{=} (\gamma, 2r).Q_2, & Q_2 &\stackrel{def}{=} (\alpha, s).Q_1 \end{aligned}$$

We now move to the semantics of FEPA, starting from two quantities specifying local states' dynamics, independently from their possible interaction with other local states.

Definition 2 (Apparent and Total Conditional Rate). *Let \mathcal{M} be a FEPA model, $P \in \mathcal{B}(\mathcal{M})$, $B \subseteq \mathcal{B}(\mathcal{M})$ and $\alpha \in \mathcal{A}$. The α -apparent rate of P and the total α -conditional transition rate from P to B are defined, respectively, as*

$$r_\alpha(P) \triangleq \sum_{P \xrightarrow{(\alpha, r)} P' \in out(P)} r \qquad q[P, B, \alpha] \triangleq \sum_{P' \in B} \sum_{P \xrightarrow{(\alpha, r)} P' \in out(P)} r$$

The α -apparent rate of a local state P can be understood as a normalized capacity, i.e., the capacity at which a unitary concentration of P -processes performs α -transitions. The total α -conditional transition rate restricts the former

with respect to a set of target local states; e.g., for \mathcal{M}_F of Example 1 we have $r_\beta(P_1) = 2r$, and $q[P_1, \{P_2\}, \beta] = r$.

Since a fluid atom is a representative of a group of sequential components of the same type, the specification is completed by fixing the group size.

Definition 3 (Concentration Function). *Let \mathcal{M} be a FEPA model. We define an initial population function for \mathcal{M} as $\nu_0 : \mathcal{B}(\mathcal{M}) \rightarrow \mathbb{N}_0$, and a concentration function for \mathcal{M} as $\nu : \mathcal{B}(\mathcal{M}) \rightarrow \mathbb{R}_{\geq 0}$.*

Definition 4 (Population-dependent Apparent Rate). *Let \mathcal{M} be a FEPA model, ν a concentration function, and $\alpha \in \mathcal{A}$. The apparent rate of α in \mathcal{M} with respect to ν is*

$$r_\alpha(\mathcal{M}_1 \parallel_L^{\mathcal{H}} \mathcal{M}_2, \nu) \triangleq \begin{cases} \mathcal{H}(r_\alpha(\mathcal{M}_1, \nu), r_\alpha(\mathcal{M}_2, \nu)), & \text{if } \alpha \in L, \\ r_\alpha(\mathcal{M}_1, \nu) + r_\alpha(\mathcal{M}_2, \nu), & \text{if } \alpha \notin L, \end{cases}$$

$$r_\alpha(P, \nu) \triangleq \sum_{P' \in \mathcal{B}(P)} \nu_{P'} \cdot r_\alpha(P').$$

The α -apparent rate in \mathcal{M} is the total rate at which α can be performed, for some ν . It is affected by synchronisations, e.g., in \mathcal{M}_F of Example 1 we have $r_\alpha(\mathcal{M}_F, \nu) = \min(s\nu_{P_2} + s\nu_{P_3}, s\nu_{Q_2})$, or $r_\alpha(\mathcal{M}_F, \nu) = (s\nu_{P_2} + s\nu_{P_3})s\nu_{Q_2}$, depending on the chosen synchronisation function \mathcal{H} . The α -apparent rate in \mathcal{M} is intended as the overall speed at which α is performed in the model; e.g., it is zero if ν_{Q_2} is zero, capturing the blocking effect of synchronisation for both choices of \mathcal{H} .

Definition 5 (Model Influence). *Let \mathcal{M} be a FEPA model, ν a concentration function for \mathcal{M} , $\alpha \in \mathcal{A}$, and $P \in \mathcal{B}(\mathcal{M})$. The model influence on P due to α in \mathcal{M} is defined as*

$$\mathcal{F}_\alpha(\mathcal{M}_1 \parallel_L^{\mathcal{H}} \mathcal{M}_2, \nu, P) \triangleq \begin{cases} \mathcal{F}_\alpha(\mathcal{M}_i, \nu, P) \frac{r_\alpha(\mathcal{M}_1 \parallel_L^{\mathcal{H}} \mathcal{M}_2, \nu)}{r_\alpha(\mathcal{M}_i, \nu)}, & \text{if } P \in \mathcal{B}(\mathcal{M}_i), \alpha \in L, \\ \mathcal{F}_\alpha(\mathcal{M}_i, \nu, P), & \text{if } P \in \mathcal{B}(\mathcal{M}_i), \alpha \notin L, \end{cases}$$

$$\mathcal{F}_\alpha(P, \nu, P') \triangleq \begin{cases} 1 & \text{if } P' \in \mathcal{B}(P), \\ 0 & \text{otherwise,} \end{cases}$$

where $\frac{r_\alpha(\mathcal{M}_1 \parallel_L^{\mathcal{H}} \mathcal{M}_2, \nu)}{r_\alpha(\mathcal{M}_i, \nu)}$ is defined as 0 when $r_\alpha(\mathcal{M}_i, \nu) = 0$.

Model influence captures the effect exerted by the model \mathcal{M} on the rate at which a local state P performs an action. In other words, the actual α -component rate of P in \mathcal{M} with concentration ν is given by the rate at which P would evolve on its own, i.e., $\nu_P \cdot r_\alpha(P)$, weighted by the influence of the model on it, i.e., $\mathcal{F}_\alpha(\mathcal{M}, \nu, P)$.

We are now ready to define the ODE semantics of a FEPA model.

Definition 6 (ODE Semantics). *Let \mathcal{M} be a FEPA model, $\mathcal{E} \subseteq \mathbb{R}^{\mathcal{B}(\mathcal{M})}$ and $f : \mathcal{E} \rightarrow \mathbb{R}^{\mathcal{B}(\mathcal{M})}$ the vector field whose components are defined for each $P \in \mathcal{B}(\mathcal{M})$ as:*

$$f_P(\nu) \triangleq \sum_{\alpha \in \mathcal{A}} \sum_{P' \in \mathcal{B}(\mathcal{M})} \nu_{P'} q(P', P, \alpha) \mathcal{F}_\alpha(\mathcal{M}, \nu, P') - \sum_{\alpha \in \mathcal{A}} \nu_P r_\alpha(P) \mathcal{F}_\alpha(\mathcal{M}, \nu, P)$$

The ODE system $\dot{\nu} = f(\nu)$ with initial condition ν_0 governs the evolution of ν over time.

The rate of change in the concentration of a local state P depends on the actual rate at which each local state P' performs transitions towards P , minus the actual rate at which P performs any transition. For instance, the ODEs of \mathcal{M}_F of Example 1 are:

$$\begin{aligned} \dot{\nu}_{P_1} &= s \mathcal{H}(\nu_{P_2} + \nu_{P_3}, \nu_{Q_2}) - 2r \nu_{P_1} & \dot{\nu}_{Q_1} &= s \mathcal{H}(\nu_{P_2} + \nu_{P_3}, \nu_{Q_2}) - 2r \nu_{Q_1} \\ \dot{\nu}_{P_2} &= r \nu_{P_1} - s \nu_{P_2} \frac{\mathcal{H}(\nu_{P_2} + \nu_{P_3}, \nu_{Q_2})}{\nu_{P_2} + \nu_{P_3}} & \dot{\nu}_{Q_2} &= 2r \nu_{P_1} - s \mathcal{H}(\nu_{P_2} + \nu_{P_3}, \nu_{Q_2}) \\ \dot{\nu}_{P_3} &= r \nu_{P_1} - s \nu_{P_3} \frac{\mathcal{H}(\nu_{P_2} + \nu_{P_3}, \nu_{Q_2})}{\nu_{P_2} + \nu_{P_3}} & & \end{aligned} \quad (1)$$

3 Differential Bisimulation and ODE Lumpability

The second level of the FEPA grammar defines a tree-like structure which strongly affects the ODE semantics. To take this into account in our differential bisimulation, we introduce the notion of *interface actions*, which intuitively captures all actions which affect the dynamics of a local state as a result of an interaction.

Definition 7 (Bound and Interface Actions). Let \mathcal{M} be a FEPA model, and $P \in \mathcal{B}(\mathcal{M})$. The set of bound actions of P in \mathcal{M} is defined as

$$\mathcal{D}(P, \mathcal{M}) \triangleq \begin{cases} L \cup \mathcal{D}(P, \mathcal{M}_i), & \text{if } \mathcal{M} = \mathcal{M}_1 \parallel_L^{\mathcal{H}} \mathcal{M}_2 \text{ and } P \in \mathcal{B}(\mathcal{M}_i), \\ \emptyset, & \text{otherwise.} \end{cases}$$

Also, the interface actions of P in \mathcal{M} are $\mathcal{I}(P, \mathcal{M}) \triangleq \mathcal{D}(P, \mathcal{M}) \cap \mathcal{A}(P)$. Lastly, for any $B \subseteq \mathcal{B}(\mathcal{M})$, we use $\mathcal{D}(B, \mathcal{M})$ for $\bigcup_{P \in B} \mathcal{D}(P, \mathcal{M})$, and $\mathcal{I}(B, \mathcal{M})$ for $\bigcup_{P \in B} \mathcal{I}(P, \mathcal{M})$.

The following notion of *structural interface* captures symmetries among the states of a FEPA model with respect to the rigid tree-like structure of the model.

Definition 8 (Structural Interface). Let \mathcal{M} be a FEPA model, and $P, Q \in \mathcal{B}(\mathcal{M})$. Then P and Q have the same structural interface in \mathcal{M} , written $P \stackrel{s.i.}{=}_{\mathcal{M}} Q$, iff

- (i) $\mathcal{A}(P) = \mathcal{A}(Q)$, and
- (ii) if there exists an $\overline{\mathcal{M}} = \mathcal{M}_1 \parallel_L^{\mathcal{H}} \mathcal{M}_2$ within \mathcal{M} with $P \in \mathcal{B}(\mathcal{M}_1)$, and $Q \in \mathcal{B}(\mathcal{M}_2)$ (or vice versa), then $\mathcal{I}(P, \overline{\mathcal{M}}) = \mathcal{I}(Q, \overline{\mathcal{M}}) = \emptyset$.

Proposition 1. For \mathcal{M} a FEPA model, $\stackrel{s.i.}{=}_{\mathcal{M}}$ is an equivalence relation.¹

¹ All proofs are provided in the extended technical report [15].

Considering Example 1 we have $D(P_1, \mathcal{M}_F) = D(P_2, \mathcal{M}_F) = \{\alpha\}$, $\mathcal{I}(P_1, \mathcal{M}_F) = \emptyset$, and $\mathcal{I}(P_2, \mathcal{M}_F) = \{\alpha\}$. Also, we have $P_2 \stackrel{s.i.}{=}_{\mathcal{M}_F} P_3$, $P_3 \not\stackrel{s.i.}{=}_{\mathcal{M}_F} Q_2$, and $P_2 \not\stackrel{s.i.}{=}_{\mathcal{M}_F} Q_2$ (capturing, for instance, that α is used by P_2 and Q_2 to interact in a specific fashion).

We can now provide the notion of differential bisimulation for FEPA models.

Definition 9 (Differential Bisimulation). *Let \mathcal{M} be a FEPA model, \mathcal{R} an equivalence relation over $\mathcal{B}(\mathcal{M})$, and $\mathcal{P} = \mathcal{B}(\mathcal{M})/\mathcal{R}$. We say that \mathcal{R} is a differential bisimulation for \mathcal{M} (DB) iff for all $(P, P') \in \mathcal{R}$ and $\alpha \in \mathcal{A}$ we have:*

- (i) $q[P, B, \alpha] = q[P', B, \alpha]$, for all $B \in \mathcal{P}$,
- (ii) $P \stackrel{s.i.}{=}_{\mathcal{M}} P'$.

We define differential bisimilarity for \mathcal{M} , denoted by $\dot{\sim}$, as the union of all DBs for \mathcal{M} , and we say that $P, P' \in \mathcal{B}(\mathcal{M})$ are differential bisimilar iff $s \dot{\sim} s'$.

As usual, we are interested in the largest differential bisimulation. We now show that differential bisimilarity is a DB, and thus it is the largest one. To do this, we prove that the transitive closure of the union of DBs is a differential bisimulation.

Proposition 2. *Let \mathcal{M} be a FEPA model, I be a set of indices, and \mathcal{R}_i a DB for \mathcal{M} , for all $i \in I$. The transitive closure of their union $\mathcal{R} = (\bigcup_{i \in I} \mathcal{R}_i)^*$ is a DB for \mathcal{M} .*

The next theorem states that DB is preserved under composition of FEPA models.

Theorem 1 (Differential Bisimulation is a Congruence). *Let $\mathcal{M}_1, \mathcal{M}_2$ be two FEPA models, and $\mathcal{R}_1, \mathcal{R}_2$ be two differential bisimulations for \mathcal{M}_1 and \mathcal{M}_2 , respectively. Then $\mathcal{R}_1 \cup \mathcal{R}_2$ is a differential bisimulation for $\mathcal{M}_1 \parallel_L^{\mathcal{H}} \mathcal{M}_2$, for any $L \subseteq \mathcal{A}$.*

Remark 1. An interesting connection between differential bisimulation and its Markovian analogues, like *Markovian bisimulation* [13] and PEPA’s *strong equivalence* [14] arises: condition (i) of DB corresponds to the condition required by Markovian bisimulation and by strong equivalence. However, in the Markovian cases states of the underlying labelled transition system (semantic elements) are related, while DB relates the states of the fluid atoms (syntactic elements). This requires to explicitly treat the influence exerted by the model on each local state (condition(ii)). Such information is instead implicitly present in the transition systems considered in the Markovian cases.

We now show that DB induces an ODE aggregation in the sense of the theory of ODE lumpability (e.g., [23]). We first exemplify it considering Example 1, for which it can be shown that $P_2 \dot{\sim} P_3$. Using the variable renaming $\nu_{P_{23}} = \nu_{P_2} + \nu_{P_3}$, by the linearity of the differential operator we can aggregate Eq. 1 as

$$\begin{aligned} \dot{\nu}_{P_1} &= s \mathcal{H}(\nu_{P_{23}}, \nu_{Q_2}) - 2r \nu_{P_1} & \dot{\nu}_{Q_1} &= s \mathcal{H}(\nu_{P_{23}}, \nu_{Q_2}) - 2r \nu_{Q_1} \\ \dot{\nu}_{P_{23}} &= 2r \nu_{P_1} - s \mathcal{H}(\nu_{P_{23}}, \nu_{Q_2}) & \dot{\nu}_{Q_2} &= 2r \nu_{P_1} - s \mathcal{H}(\nu_{P_{23}}, \nu_{Q_2}) \end{aligned}$$

If the initial conditions are such that $\nu_{0P_{23}} = \nu_{0P_2} + \nu_{0P_3}$, the solutions satisfy $\nu_{P_{23}}(t) = \nu_{P_2}(t) + \nu_{P_3}(t)$ for all t . As discussed, this is analogous to ordinary lumpability in CTMCs, where the probability of being in a state of the aggregated chain is equal to the sum of the probabilities of being in the states of the related equivalence class [3].

Noteworthy, condition(i) of DB does not capture ODE aggregation if ignoring structural interface. Assuming $\beta = \gamma$, $\{\{P_1, Q_1\}, \{P_2, P_3, Q_2\}\}$ satisfies condition(i). Yet, $P_2 \not\stackrel{s.i.}{\sim}_{\mathcal{M}_{\mathbb{F}}} Q_2$ and $P_3 \not\stackrel{s.i.}{\sim}_{\mathcal{M}_{\mathbb{F}}} Q_2$. This results in ODEs with nonlinear terms in ν_{P_2} and ν_{Q_2} , such as $\mathcal{H}(\nu_{P_2} + \nu_{P_3}, \nu_{Q_2})$, which cannot be written in terms of $\nu_{P_2} + \nu_{Q_2}$.

We formalize such ODE aggregation in terms of ODE lumpability by an aggregation matrix. Given a FEPA model \mathcal{M} and a partition \mathcal{P} of $\mathcal{B}(\mathcal{M})$, the *aggregation matrix* of \mathcal{P} has $|\mathcal{P}| \times |\mathcal{B}(\mathcal{M})|$ components given as $(M_{\mathcal{P}})_{i,j} = 1$ if $P_j \in B_i$, and $(M_{\mathcal{P}})_{i,j} = 0$ otherwise, where $B_i \in \mathcal{P}$ and $P_j \in \mathcal{B}(\mathcal{M})$, with $i \in \{1, \dots, |\mathcal{P}|\}$ and $j \in \{1, \dots, |\mathcal{B}(\mathcal{M})|\}$.

Definition 10 (ODE Lumpability). *Let \mathcal{M} be a FEPA model, f its vector field, and \mathcal{P} a partition of $\mathcal{B}(\mathcal{M})$. The ODE system $\dot{\nu} = f(\nu)$ is lumpable by $M_{\mathcal{P}}$ if and only if*

$$M_{\mathcal{P}}f(\nu) = M_{\mathcal{P}}f(\overline{M}_{\mathcal{P}}M_{\mathcal{P}}\nu), \quad \text{for all } \nu, \tag{2}$$

where $\overline{M}_{\mathcal{P}}$ is any generalized right inverse of $M_{\mathcal{P}}$, i.e., a matrix satisfying $M_{\mathcal{P}}\overline{M}_{\mathcal{P}} = \mathbb{I}$.

The vector ν has $|\mathcal{B}(\mathcal{M})|$ components, each being the concentration of a local state of \mathcal{M} at a certain time. For \mathcal{P} a partition of $\mathcal{B}(\mathcal{M})$, $M_{\mathcal{P}}\nu$ has $|\mathcal{P}|$ components, each equal to the sum of the components of ν in the corresponding block. The vector $\overline{M}_{\mathcal{P}}M_{\mathcal{P}}\nu$ has again $|\mathcal{B}(\mathcal{M})|$ components, obtained by first summing the components of ν in each block ($M_{\mathcal{P}}\nu$) and subsequently redistributing it to the local states of the block. Equation 2 demands that the sum of the dynamics of local states of a block, i.e., $M_{\mathcal{P}}f(\nu)$, can be expressed as a function of the aggregated vector, i.e., $M_{\mathcal{P}}\nu$, only.

Theorem 2 (Differential Bisimulation and Lumpability). *Let \mathcal{M} be a FEPA model, \mathcal{R} a differential bisimulation, and $\mathcal{P} = \mathcal{B}(\mathcal{M})/\mathcal{R}$. The ODEs of \mathcal{M} are lumpable by $M_{\mathcal{P}}$.*

Proof (sketch). We have to show that Eq. 2 holds. The proof uses Proposition 4 and Lemma 4 given in [15], and here discussed. For ν a concentration function for \mathcal{M} and \mathcal{P} a partition of $\mathcal{B}(\mathcal{M})$, we define $[\nu]^{\mathcal{P}}$, the \mathcal{P} -redistribution of ν , as

$$[\nu]^{\mathcal{P}} = \overline{M}_{\mathcal{P}}M_{\mathcal{P}}\nu. \tag{3}$$

Thus, we have to show that for any ν it holds $M_{\mathcal{P}}f(\nu) = M_{\mathcal{P}}f([\nu]^{\mathcal{P}})$. Recalling the definition of the aggregation matrix $M_{\mathcal{P}}$, it is enough to show that for any $B \in \mathcal{P}$ and ν

$$\sum_{P \in B} f_P(\nu) = \sum_{P \in B} f_P(\overline{M}_{\mathcal{P}}M_{\mathcal{P}}\nu) = \sum_{P \in B} f_P([\nu]^{\mathcal{P}}),$$

i.e., we verify Eq. 2 componentwise. Summing over $P \in B$ both sides of f of Definition 6, and using that $\sum_{P \in B} q(P', P, \alpha) = q[P', B, \alpha]$, as well as a decomposition of the sum over states, i.e., $\sum_{P' \in \mathcal{B}(\mathcal{M})}(\cdot) = \sum_{B \in \mathcal{P}} \sum_{P' \in B}(\cdot)$, we obtain

$$\begin{aligned}
 \sum_{P \in B} f_P(\nu) &= \sum_{\alpha \in \mathcal{A}} \sum_{P' \in \mathcal{B}(\mathcal{M})} \nu_{P'} q[P', B, \alpha] \mathcal{F}_\alpha(\mathcal{M}, \nu, P') \\
 &\quad - \sum_{P \in B} \nu_P \sum_{\tilde{B} \in \mathcal{P}} \sum_{\alpha \in \mathcal{A}} q[P, \tilde{B}, \alpha] \mathcal{F}_\alpha(\mathcal{M}, \nu, P) \\
 &= \sum_{\tilde{B} \in \mathcal{P}} \sum_{P' \in \tilde{B}} \sum_{\alpha \in \mathcal{A}} q[P', B, \alpha] \mathcal{F}_\alpha(\mathcal{M}, \nu, P') \nu_{P'} \\
 &\quad - \sum_{P \in B} \sum_{\tilde{B} \in \mathcal{P}} \sum_{\alpha \in \mathcal{A}} q[P, \tilde{B}, \alpha] \mathcal{F}_\alpha(\mathcal{M}, \nu, P) \nu_P
 \end{aligned} \tag{4}$$

We are left with showing that for any ν Eq. 4 does not change if we replace ν with $[\nu]^\mathcal{P}$. This corresponds to saying that it can be expressed as a function of the sums of the concentrations in each block of \mathcal{P} only. For any P and any $B \in \mathcal{P}$ we can write $\sum_{\alpha \in \mathcal{A}} q[P, B, \alpha] \mathcal{F}_\alpha(\mathcal{M}, \nu, P) = \sum_{\alpha \in \mathcal{A}(P)} q[P, B, \alpha] \mathcal{F}_\alpha(\mathcal{M}, \nu, P)$, which follows from observing that any $\alpha \notin \mathcal{A}(P)$ brings 0-contribution to the equation (because $\alpha \notin \mathcal{A}(P) \implies r_\alpha(P) = 0 \implies q[P, B, \alpha] = 0, \forall B$). We now exploit the fact that \mathcal{P} is induced by a DB on $\mathcal{B}(\mathcal{M})$, as sketched below in the following three points.

- (i) We have that for all $B \in \mathcal{P}$, for all $Q, Q' \in B$, $q[Q, \tilde{B}, \alpha] = q[Q', \tilde{B}, \alpha]$ for all $\tilde{B} \in \mathcal{P}$ and all $\alpha \in \mathcal{A}$, which, in turn, implies $\mathcal{A}(Q) = \mathcal{A}(Q')$.
- (ii) We show, in Proposition 4, that for all $B \in \mathcal{P}$, and all $Q, Q' \in B$, $\mathcal{F}_\alpha(\mathcal{M}, \nu, Q) = \mathcal{F}_\alpha(\mathcal{M}, \nu, Q')$ for all ν and all $\alpha \in \mathcal{A}(Q) = \mathcal{A}(Q')$. Thus, it holds that for all $B, \tilde{B} \in \mathcal{P}$, all $P, P' \in B$, and all ν , $\sum_{\alpha \in \mathcal{A}} q[P, \tilde{B}, \alpha] \mathcal{F}_\alpha(\mathcal{M}, \nu, P) = \sum_{\alpha \in \mathcal{A}} q[P', \tilde{B}, \alpha] \mathcal{F}_\alpha(\mathcal{M}, \nu, P')$. That is, the summation is equal for all local states of block B . Proposition 4 establishes a relation between structural interface Definition 8 and model influence Definition 5, essentially saying that if two local states have the same structural interface within a model, then they receive the same influence from the model.
- (iii) We show, in Lemma 4, that for any $P \in \mathcal{B}(\mathcal{M})$, α and ν it holds $\mathcal{F}_\alpha(\mathcal{M}, \nu, P) = \mathcal{F}_\alpha(\mathcal{M}, [\nu]^\mathcal{P}, P)$. This is used to infer that for any $B, \tilde{B} \in \mathcal{P}$, any $P \in B$ and any ν :

$$\sum_{\alpha \in \mathcal{A}} q[P, \tilde{B}, \alpha] \mathcal{F}_\alpha(\mathcal{M}, \nu, P) = \sum_{\alpha \in \mathcal{A}} q[P, \tilde{B}, \alpha] \mathcal{F}_\alpha(\mathcal{M}, [\nu]^\mathcal{P}, P).$$

That is, the summation can be expressed as a function of the sums of the concentration in each block of \mathcal{P} . In other words, the model influence received by a local state depends on the concentration of the other local states only through the sum of the concentrations within blocks of \mathcal{P} , thus a change in the concentrations which preserves the total concentrations of each block does not affect the model influence.

```

1 DifferentialBisimilarity( $\mathcal{M}, \mathcal{P}$ ) :=
2   RefineSI( $\mathcal{M}, \mathcal{P}$ ) //Refine  $\mathcal{P}$  wrt condition (ii)
3   RefineQ( $\mathcal{M}, \mathcal{P}$ ) //Iteratively refines  $\mathcal{P}$  wrt condition (i)
4   RefineSI( $\mathcal{M}, \mathcal{P}$ ) :=
5   for all ( $\alpha \in \mathcal{A}(\mathcal{M})$ )
6     refineAccordingToComp( $\alpha, \mathcal{P}$ ) //Refine  $\mathcal{P}$  wrt comp[ $\alpha$ ], for all  $\alpha$ 
7
8   RefineQ( $\mathcal{M}, \mathcal{P}$ ) :=
9   SplS =  $\mathcal{A}(\mathcal{M}) \times \mathcal{P}$  //All ( $\alpha, B$ ) are considered as candidate
// splitters
10  while (SplS  $\neq \emptyset$ )
11    ( $\alpha, B_{spl}$ ) = pop(SplS) //choose and remove a candidate splitter
12    Split( $\alpha, B_{spl}, \mathcal{P}, SplS$ ) //split all blocks of  $\mathcal{P}$  wrt ( $\alpha, B_{spl}$ )
    
```

Algorithm 1. An algorithm for computing differential bisimilarity

Now that all the proof ingredients have been provided, we can rewrite Eq. 4 as follows, where we use that for any $B \in \mathcal{P}$, $\sum_{P \in B} [\nu]_P^{\mathcal{P}} = \sum_{P \in B} \nu_P$, which arises from Eq. 3 and the fact that the matrix $\overline{M}_{\mathcal{P}}$ must satisfy $M_{\mathcal{P}} \overline{M}_{\mathcal{P}} = \mathbb{I}$:

$$\begin{aligned}
 \sum_{P \in B} f_P(\nu) &= \sum_{\tilde{B} \in \mathcal{P}} \sum_{\alpha \in \mathcal{A}} q[P', B, \alpha] \mathcal{F}_{\alpha}(\mathcal{M}, \nu, P') \sum_{P' \in \tilde{B}} \nu_{P'} \\
 &\quad - \sum_{\tilde{B} \in \mathcal{P}} \sum_{\alpha \in \mathcal{A}} q[P, \tilde{B}, \alpha] \mathcal{F}_{\alpha}(\mathcal{M}, \nu, P) \sum_{P \in B} \nu_P \\
 &= \sum_{\tilde{B} \in \mathcal{P}} \sum_{\alpha \in \mathcal{A}} q[P', B, \alpha] \mathcal{F}_{\alpha}(\mathcal{M}, [\nu]^{\mathcal{P}}, P') \sum_{P' \in \tilde{B}} [\nu]_{P'}^{\mathcal{P}} \\
 &\quad - \sum_{\tilde{B} \in \mathcal{P}} \sum_{\alpha \in \mathcal{A}} q[P, \tilde{B}, \alpha] \mathcal{F}_{\alpha}(\mathcal{M}, [\nu]^{\mathcal{P}}, P) \sum_{P \in B} [\nu]_P^{\mathcal{P}} = \sum_{P \in B} f_P([\nu]^{\mathcal{P}})
 \end{aligned}$$

□

4 Computing Differential Bisimilarity

We now provide an efficient algorithm for computing differential bisimilarity obtained by extending and reusing well-known partition refinement algorithms, e.g. [1, 13, 20].

In order to apply partition refinement to differential bisimilarity, let us first note that condition (ii) of DB can be dealt with as an initialization step that pre-partitions the local states according to their structural interface. Instead, condition (i) requires the usual partition-refinement treatment: starting from the partition obtained after initialization, the blocks are iteratively *split* until there exists a block and an action (i.e., a candidate splitter) for which condition (i) does not hold. The algorithm takes in input any initial partition \mathcal{P} , useful e.g. to specify local states that should not be equated, and terminates giving the largest differential bisimilarity which refines \mathcal{P} for the considered model.

Overview. **DifferentialBisimilarity**, our algorithm, is given in Algorithm 1, where \mathcal{M} is the input FEPA model and \mathcal{P} the initial partition. We use $\mathcal{A}(\mathcal{M})$ for the set of actions in \mathcal{M} , and $\mathcal{T}(\mathcal{M}) \triangleq \{P' \xrightarrow{(\alpha, r)} P'' \in \text{out}(P') \mid P' \in \mathcal{B}(\mathcal{M})\}$ for its multi-set of transitions. Note that $|\mathcal{A}(\mathcal{M})| \leq |\mathcal{T}(\mathcal{M})|$. Also, we use $t_{\mathcal{M}}$ for $|\mathcal{T}(\mathcal{M})|$, and $s_{\mathcal{M}}$ for $|\mathcal{B}(\mathcal{M})|$, and we do not distinguish an equivalence relation from its induced partition. **RefineSI** implements the initialization step, yielding the coarsest refinement of \mathcal{P} with respect to condition (ii). **RefineQ** iteratively computes the coarsest refinement satisfying condition (i). Overall, the algorithm is correct, as the iterative refinements preserve condition (ii). It is assumed that \mathcal{M} is stored as the list $\mathcal{T}(\mathcal{M})$, requiring $O(t_{\mathcal{M}})$ space. In order to represent partitions \mathcal{P} , $\mathcal{B}(\mathcal{M})$ is stored as a list, while a block of \mathcal{P} is a list of pointers to its states, requiring in total $O(t_{\mathcal{M}} + s_{\mathcal{M}})$ to store \mathcal{M} .

RefineSI. This procedure is based on a simple rephrasing of Definition 8: given a FEPA model \mathcal{M} and $P_1, P_2 \in \mathcal{B}(\mathcal{M})$ with $\mathcal{A}(P_1) = \mathcal{A}(P_2)$, we have $P_1 \stackrel{s.i.}{\sim} P_2$ if and only if for all $\alpha \in \mathcal{A}(P_1)$ and for all occurrences $\overline{\mathcal{M}} = \mathcal{M}_1 \parallel_L \mathcal{M}_2$ within \mathcal{M} with $\alpha \in L \cap \mathcal{A}(P_1)$ we have that P_1 and P_2 either belong to the same \mathcal{M}_i , or do not belong to any of the two (i.e., $P_1, P_2 \notin \mathcal{B}(\overline{\mathcal{M}})$). Also, if two states have the same innermost compositional operator binding α , then they share all outers too. No further information is required about compositional operators, and thus we assume that each $P \in \mathcal{B}(\mathcal{M})$ has a list *comp* containing an entry per action in $\mathcal{A}(P)$, each being a triple storing the action, the (identifier of the) innermost compositional operator affecting P and binding the key action, and the side of the operator to which P belongs. Also, each *comp* is assumed to be sorted with respect to a total ordering on \mathcal{A} . We use $\text{comp}[\alpha]$ for the values associated with α in *comp*. For instance, for $\mathcal{M} = \mathcal{M}_1 \parallel_L \mathcal{M}_2$, id^* the identifier of \parallel_L , $P_1 \in \mathcal{B}(\mathcal{M}_1)$ and $\alpha \in \mathcal{A}(P_1) \cap L$, we have $P_1.\text{comp}[\alpha] = (\text{id}^*, \text{left})$ if no further compositional operators binding α appear in the syntax-tree path leading to P_1 . All *comp* require $O(t_{\mathcal{M}})$ space in total: each $P.\text{comp}$ has at most one entry per transition with source P , and thus at most $t_{\mathcal{M}}$ entries appear in all *comp*. By defining a total ordering on *comp*'s values, **RefineSI** reduces to iteratively sorting all $P \in \mathcal{B}(\mathcal{M})$ according to $P.\text{comp}[\alpha]$ for all $\alpha \in \mathcal{A}(\mathcal{M})$ (Line 6).² The sorting for each α can be performed in $O(s_{\mathcal{M}} \cdot \log s_{\mathcal{M}})$, and if we scan $\mathcal{A}(\mathcal{M})$ according to the ordering of \mathcal{A} we can access the elements of the lists in constant time, requiring $O(t_{\mathcal{M}} \cdot s_{\mathcal{M}} \cdot \log s_{\mathcal{M}})$ time to perform the sorting. Overall, this yields $O(t_{\mathcal{M}} \cdot s_{\mathcal{M}} \cdot \log s_{\mathcal{M}})$ time complexity.

Theorem 3. *Let \mathcal{M} be a FEPA model and \mathcal{P} a partition of $\mathcal{B}(\mathcal{M})$. **RefineSI** computes the coarsest refinement of \mathcal{P} satisfying condition (ii). It can be implemented with time and space complexities $O(t_{\mathcal{M}} \cdot s_{\mathcal{M}} \cdot \log s_{\mathcal{M}})$ and $O(t_{\mathcal{M}} + s_{\mathcal{M}})$, respectively.*

RefineQ. Condition (i) ignores compositional operators. Thus, **RefineQ** treats \mathcal{M} as a *stochastic labeled transition system* (STLS), i.e. a transition system

² $P.\text{comp}[\alpha]$ is *nil* if $\alpha \notin \mathcal{A}(P)$, and *free* if $\alpha \in \mathcal{A}(P)$ and $\alpha \notin \mathcal{D}(P, \mathcal{M})$, so to tell apart states performing different actions.

(with a root per fluid atom) where transitions are labeled by an action and a real. This allows us to use the algorithm for *Markovian bisimilarity* of SLTSs presented in [9, 13]. In fact, as discussed, condition (i) corresponds to Markovian bisimulation. Indeed, `RefineQ` is a straightforward rephrasing of the algorithm of [9, 13] to FEPA notation. An in-depth discussion of the algorithm can be found in [9, 13], while we hereby give a high-level description. We start recalling the algorithm's complexities.

Theorem 4 (Adapted from [13]). *For \mathcal{M} a FEPA model and \mathcal{P} a partition of $\mathcal{B}(\mathcal{M})$, `RefineQ` gives the coarsest refinement of \mathcal{P} satisfying condition (i) of DB. It can be realized with time and space complexities $O(t_{\mathcal{M}} \cdot \log s_{\mathcal{M}})$ and $O(t_{\mathcal{M}} + s_{\mathcal{M}})$, respectively.*

Refinements are based on *splitters* (α, B_{spl}) , with $\alpha \in \mathcal{A}(\mathcal{M})$ and $B_{spl} \in \mathcal{P}$: a block $B \in \mathcal{P}$ is split with respect to (α, B_{spl}) in disjoint sub-blocks, each containing states with same total α -conditional transition rate towards B_{spl} . `RefineQ` starts (Line 9) generating a set `Spls` of initial potential splitters (α, B) for each $\alpha \in \mathcal{A}(\mathcal{M})$ and $B \in \mathcal{P}$. Then, Lines 10–12 iterate until there are potential splitters to be considered: a splitter is selected and removed from `Spls`, and the procedure `Split` is invoked to refine each block of \mathcal{P} according to the selected splitter, and to generate new candidate splitters. Due to space constraints we do not detail the `Split` procedure.

Summary. Theorems 3 and 4 allow us to conclude that `DifferentialBisimilarity` has time and space complexities $O(t_{\mathcal{M}} \cdot s_{\mathcal{M}} \cdot \log s_{\mathcal{M}})$ and $O(t_{\mathcal{M}} + s_{\mathcal{M}})$, respectively.

5 Related Work

The *label equivalence* presented in [26] captures *exact fluid lumpability*, a different notion of ODE lumpability than the one captured by DB, where processes are equivalent whenever their ODE solutions are equal at all time points, provided they have same initial conditions. Label equivalence works at a coarser level of granularity than DB, as it relates whole fluid atoms, and not their individual local states, essentially requiring an isomorphism between them. Further, the conditions for equivalence in [26] include universal quantifiers over the uncountable set of concentration functions which are difficult to check automatically. Indeed, no algorithm for computing the coarsest partition was developed for label equivalence. In contrast, DB is given in terms of syntactic elements only, allowing us to provide an efficient algorithm to compute the largest one of a model. In [25] the same authors extended the framework of [26] to the notion of ODE lumpability considered in this paper, for which, however, the same limitations as those of label equivalence apply.

The relationship between formal languages and ODEs induced by their semantics has been studied also in other contexts, with complementary approaches. In [7] it is presented a model-order reduction technique for κ [8],

a rule-based language for chemical systems representing bindings between molecules in an explicit graph-based way. The aggregation method, called *fragmentation*, identifies a linear transformation of the state space yielding a subspace with a closed dynamics, i.e., whose ODEs depend only on the variables of that subspace. This may give an *improper lumping* (see [19]), as the same state may appear in more than one aggregate, and thus it is not necessarily induced by a partition of the state space. More practically, it can be shown that \mathcal{M}_F of Example 1 can be encoded in κ in case $\mathcal{H} = \cdot$, but it is not reduced by fragmentation. (Dually, there exist κ 's models which can be encoded in FEPA that are reduced by fragmentation but not by DB). However, clearly, the two target languages are different; κ is based on the *law of mass action*, where the rate of interaction is proportional to the product of the participants' concentrations, similarly to FEPA's $\mathcal{H} = \cdot$. Instead, FEPA is process-based, with the rule of interaction implicit in the rigid compositional structure, while a chemical system is an unstructured set of interacting species. Also, FEPA allows for a synchronisation semantics based on capacity-sharing arguments (in the case $\mathcal{H} = \min$).

More closely related is the bisimulation in [5], which induces both ODE lumpabilities of Definition 10 and [26]. The difference is again in the language-specific definitions of equivalence. While DB is a relation over process algebra terms, in [5] symmetries are exploited between binding sites of κ agents. Also, [5] requires stronger symmetries than DB, as the latter considers those specific to the notion of lumpability of Definition 10 only. For example, it can be shown that the DB $\{\{P_1\}, \{P_2, P_3\}, \{Q_1\}, \{Q_2\}\}$ of \mathcal{M}_F of Example 1 does not satisfy the notion of lumpability of [26].

The combination of the notion of bisimulation and ODEs has been explored also by the control theory community, most notably in the work of Pappas and co-authors (e.g., [10, 21]) and van der Schaft [22]. However, the setting is different. When studied for model reduction, they essentially deal with a state space representation with an explicit output map, e.g., the matrix C in the linear dynamical system $\dot{x} = Ax + Bu$, $y = Cx$. A bisimulation is thus related to unobservability subspaces (cf. [21, Sect. 8.1] and [22, Corollary 6.4]). By contrast, in this paper we work with a nonlinear system in the form $\dot{x} = A(x)$ (with A a nonlinear vector field) where bisimulation is related to aggregation; in the aggregated model only a linear combination of the original state space variables can be recovered. More in general, the bisimulations in [10, 21, 22] are defined directly at the level of the dynamical system (either in discrete or continuous time) whereas DB is defined at the language level, as a relation between process terms.

6 Conclusion

We presented differential bisimulation, a behavioral relation for process calculi with ordinary differential equation (ODE) semantics. This study follows the line of research on equivalence relations for quantitative models of computation. In particular, differential bisimulation is defined as a relation over a discrete set of

process terms inducing an aggregation of the ODEs, analogously to Markovian bisimulations for process calculi which lead to the lumping of the underlying Markov process. Differential bisimulation allows relating local states of somewhat *heterogenous* processes instead of essentially isomorphic ones, as required in previous work. In addition, it is given in terms of syntactic conditions and it does not involve universal quantifiers over the expressions determining the ODE system. This, together with a conceptual similarity with Markovian bisimulations, allowed for the development of a partition-refinement algorithm for computing differential bisimilarity, largely reusing available results in the Markovian setting. As with its Markovian counterparts, differential bisimulation provides only sufficient conditions for ODE lumping. In this respect, an interesting line of investigation will be how to relax the current assumptions to obtain coarser aggregations. Another interesting problem is whether differential bisimulation implies lumpability also of the underlying Markov chain obtained when considering a Markovian semantics.

References

1. Baier, C., Engelen, B., Majster-Cederbaum, M.E.: Deciding bisimilarity and similarity for probabilistic processes. *J. Comput. Syst. Sci.* **60**(1), 187–231 (2000)
2. Bernardo, M.: A survey of Markovian behavioral equivalences. In: Bernardo, M., Hillston, J. (eds.) *SFM 2007*. LNCS, vol. 4486, pp. 180–219. Springer, Heidelberg (2007)
3. Buchholz, P.: Exact and ordinary lumpability in finite Markov chains. *J. Appl. Probab.* **31**(1), 59–75 (1994)
4. Buchholz, P.: Markovian process algebra: composition and equivalence. In: *Proceedings of 2nd PAPM Workshop*. Erlangen, Germany (1994)
5. Camporesi, F., Feret, J.: Formal reduction for rule-based models. *ENTCS* **276**, 29–59 (2011)
6. Ciocchetta, F., Hillston, J.: Bio-PEPA: a framework for the modelling and analysis of biological systems. *TCS* **410**(33–34), 3065–3084 (2009)
7. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Abstracting the differential semantics of rule-based models: exact and automated model reduction. In: *LICS*, pp. 362–381 (2010)
8. Danos, V., Laneve, C.: Formal molecular biology. *TCS* **325**(1), 69–110 (2004)
9. Derisavi, S., Hermanns, H., Sanders, W.H.: Optimal state-space lumping in Markov chains. *Inf. Process. Lett.* **87**(6), 309–315 (2003)
10. Haghverdi, E., Tabuada, P., Pappas, G.J.: Bisimulation relations for dynamical, control, and hybrid systems. *TCS* **342**(2–3), 229–261 (2005)
11. Hayden, R.A., Bradley, J.T.: A fluid analysis framework for a Markovian process algebra. *TCS* **411**(22–24), 2260–2297 (2010)
12. Hermanns, H., Rettelbach, M.: Syntax, semantics, equivalences, and axioms for MTIPP. In: *Proceedings of Process Algebra and Probabilistic Methods*, pp. 71–87. Erlangen (1994)
13. Hermanns, H., Siegle, M.: Bisimulation algorithms for stochastic process algebras and their BDD-based implementation. In: Katoen, J.-P. (ed.) *AMAST-ARTS 1999*, *ARTS 1999*, and *AMAST-WS 1999*. LNCS, vol. 1601, p. 244. Springer, Heidelberg (1999)

14. Hillston, J.: *A Compositional Approach to Performance Modelling*, CUP. Cambridge University Press, New York (1996)
15. Iacobelli, G., Tribastone, M., Vandin, A.: Differential bisimulation for a Markovian process algebra. Extended Version. QUANTICOL TR-QC-04-2015 (2015). <http://milner.inf.ed.ac.uk/wiki/files/W232G9A7/mfcs2015ExtendedTRpdf.html>
16. Kurtz, T.G.: Solutions of ordinary differential equations as limits of pure jump Markov processes. *J. Appl. Probab.* **7**(1), 49–58 (1970)
17. Kurtz, T.G.: *Approximation of Population Processes*, vol. 36. SIAM, Philadelphia (1981)
18. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. *Inf. Comput.* **94**(1), 1–28 (1991)
19. Okino, M.S., Mavrouniotis, M.L.: Simplification of mathematical models of chemical reaction systems. *Chem. Rev.* **2**(98), 391–408 (1998)
20. Paige, R., Tarjan, R.: Three partition refinement algorithms. *SIAM* **16**(6), 973–989 (1987)
21. Pappas, G.J.: Bisimilar linear systems. *Automatica* **39**(12), 2035–2047 (2003)
22. van der Schaft, A.J.: Equivalence of dynamical systems by bisimulation. *IEEE TAC* **49**, 2160–2172 (2004)
23. Toth, J., Li, G., Rabitz, H., Tomlin, A.S.: The effect of lumping and expanding on kinetic differential equations. *SIAM J. Appl. Math.* **57**(6), 1531–1556 (1997)
24. Tribastone, M., Gilmore, S., Hillston, J.: Scalable differential analysis of process algebra models. *IEEE TSE* **38**(1), 205–219 (2012)
25. Tschaikowski, M., Tribastone, M.: A unified framework for differential aggregations in Markovian process algebra. *JLAMP* **84**(2), 238–258 (2015)
26. Tschaikowski, M., Tribastone, M.: Exact fluid lumpability for Markovian process algebra. In: Koutny, M., Ulidowski, I. (eds.) *CONCUR 2012*. LNCS, vol. 7454, pp. 380–394. Springer, Heidelberg (2012)

On the Hardness of Almost–Sure Termination

Benjamin Lucien Kaminski^(✉) and Joost-Pieter Katoen^(✉)

Software Modeling and Verification Group,
RWTH Aachen University, Aachen, Germany
{benjamin.kaminski,katoen}@cs.rwth-aachen.de

Abstract. This paper considers the computational hardness of computing expected outcomes and deciding (universal) (positive) almost–sure termination of probabilistic programs. It is shown that computing lower and upper bounds of expected outcomes is Σ_1^0 - and Σ_2^0 -complete, respectively. Deciding (universal) almost–sure termination as well as deciding whether the expected outcome of a program equals a given rational value is shown to be Π_2^0 -complete. Finally, it is shown that deciding (universal) positive almost–sure termination is Σ_2^0 -complete (Π_3^0 -complete).

Keywords: Probabilistic programs · Expected outcomes · Almost–sure termination · Positive almost–sure termination · Computational hardness

1 Introduction

Probabilistic programs [1] are imperative programs with the ability to toss a (possibly) biased coin and proceed their execution depending on the outcome of the coin toss. They are used in randomized algorithms, in security to describe cryptographic constructions (such as randomized encryption) and security experiments [2], and in machine learning to describe distribution functions that are analyzed using Bayesian inference [3]. Probabilistic programs are typically just a small number of lines, but hard to understand and analyze, let alone algorithmically. This paper considers the computational hardness of two main analysis problems (and variations thereof) for probabilistic programs:

1. Computing expected outcomes: Is the expected outcome of a program variable smaller than, equal to, or larger than a given rational number?
2. Deciding [universal] (positive) almost–sure termination: Does a program terminate [on all inputs] with probability one (within an expected finite number of computation steps)?

The first analysis problem is related to determining weakest pre-expectations of probabilistic programs [4, 5]. Almost–sure termination is an active field of research [6]. A lot of work has been done towards automated reasoning for

This research is funded by the Excellence Initiative of the German federal and state governments and by the EU FP7 MEALS project.

almost-sure termination. For instance, [7] gives an overview of some particularly interesting examples of probabilistic logical programs and the according intuition for proving almost-sure termination. Arons *et al.* [8] reduce almost-sure termination to termination of non-deterministic programs by means of a planner. This idea has been further exploited and refined into a pattern-based approach with prototypical tool support [9].

Despite the existence of several (sometimes automated) approaches to tackle almost-sure termination, most authors claim that it must intuitively be harder than the termination problem for ordinary programs. To mention a few, Morgan [10] remarks that while partial correctness for small-scale examples is not harder to prove than for ordinary programs, the case for total correctness of a probabilistic loop must be harder to analyze. Esparza *et al.* [9] claim that almost-sure termination must be harder to decide than ordinary termination since for the latter a topological argument suffices while for the former arithmetical reasoning is needed. The computational hardness of almost-sure termination has however received scant attention. As a notable exception, [11] establishes that deciding almost-sure termination of certain concurrent probabilistic programs is in Π_2^0 .

In this paper, we give precise classifications of the level of arithmetical reasoning that is needed to decide the aforementioned analysis problems by establishing the following results: We first show that computing lower bounds on the expected outcome of a program variable v after executing a probabilistic program P on a given input η is Σ_1^0 -complete and therefore arbitrarily close approximations from below are computable. Computing upper bounds, on the other hand, is shown to be Σ_2^0 -complete, thus arbitrarily close approximations from above are not computable in general. Deciding whether an expected outcome equals some rational is shown to be Π_2^0 -complete.

For the second analysis problem—almost-sure termination—we obtain that deciding almost-sure termination of a probabilistic program P on a given input η is Π_2^0 -complete. While for ordinary programs we have a complexity leap when moving from the non-universal to the universal halting problem, we establish that *this is not the case for probabilistic programs*: Deciding universal almost-sure termination turns out to be Π_2^0 -complete, too. The case for *positive* almost-sure termination is different however: While deciding (non-universal) positive almost-sure termination is Σ_2^0 -complete, we show that universal positive almost-sure termination is Π_3^0 -complete.

2 Preliminaries

As indicated, our hardness results will be stated in terms of levels in the arithmetical hierarchy—a concept we briefly recall:

Definition 1 (Arithmetical Hierarchy [12,13]). *For every $n \in \mathbb{N}$, the class Σ_n^0 is defined as $\Sigma_n^0 = \{\mathcal{A} \mid \mathcal{A} = \{x \mid \exists y_1 \forall y_2 \exists y_3 \cdots \exists/\forall y_n: (x, y_1, y_2, y_3, \dots, y_n) \in \mathcal{R}\}, \mathcal{R} \text{ is a decidable relation}\}$, the class*

Π_n^0 is defined as $\Pi_n^0 = \{\mathcal{A} \mid \mathcal{A} = \{x \mid \forall y_1 \exists y_2 \forall y_3 \cdots \exists/\forall y_n : (x, y_1, y_2, y_3, \dots, y_n) \in \mathcal{R}\}, \mathcal{R} \text{ is a decidable relation}\}$ and the **class** Δ_n^0 is defined as $\Delta_n^0 = \Sigma_n^0 \cap \Pi_n^0$. Note that we require that the values of the variables are drawn from a recursive domain. *Multiple consecutive quantifiers of the same type* can be contracted to *one* quantifier of that type, so the number n really refers to the number of necessary *quantifier alternations* rather than to the number of quantifiers used. A set \mathcal{A} is called **arithmetical**, iff $\mathcal{A} \in \Gamma_n^0$, for $\Gamma \in \{\Sigma, \Pi, \Delta\}$ and $n \in \mathbb{N}$. The arithmetical sets form a strict hierarchy, i.e. $\Delta_n^0 \subset \Gamma_n^0 \subset \Delta_{n+1}^0$ and $\Sigma_n^0 \neq \Pi_n^0$ holds for $\Gamma \in \{\Sigma, \Pi\}$ and $n \geq 1$. Furthermore, note that $\Sigma_0^0 = \Pi_0^0 = \Delta_0^0 = \Delta_1^0$ is exactly the class of the decidable sets and Σ_1^0 is exactly the class of the recursively enumerable sets.

Next, we recall the concept of many-one reducibility and completeness:

Definition 2 (Many-One Reducibility and Completeness [13–15]). *Let \mathcal{A}, \mathcal{B} be arithmetical sets and let X be some appropriate universe such that $\mathcal{A}, \mathcal{B} \subseteq X$. \mathcal{A} is called **many-one-reducible** to \mathcal{B} , denoted $\mathcal{A} \leq_m \mathcal{B}$, iff there exists a computable function $f: X \rightarrow X$, such that $\forall x \in X: (x \in \mathcal{A} \iff f(x) \in \mathcal{B})$. If f is a function such that f many-one reduces \mathcal{A} to \mathcal{B} , we denote this by $f: \mathcal{A} \leq_m \mathcal{B}$. Note that \leq_m is transitive.*

\mathcal{A} is called Γ_n^0 -complete, for $\Gamma \in \{\Sigma, \Pi, \Delta\}$, iff both $\mathcal{A} \in \Gamma_n^0$ and \mathcal{A} is Γ_n^0 -hard, meaning $\mathcal{C} \leq_m \mathcal{A}$, for any set $\mathcal{C} \in \Gamma_n^0$. Note that if \mathcal{A} is Γ_n^0 -complete and $\mathcal{A} \leq_m \mathcal{B}$, then \mathcal{B} is necessarily Γ_n^0 -hard. Furthermore, note that if \mathcal{A} is Σ_n^0 -complete, then $\mathcal{A} \in \Sigma_n^0 \setminus \Pi_n^0$. Analogously if \mathcal{A} is Π_n^0 -complete, then $\mathcal{A} \in \Pi_n^0 \setminus \Sigma_n^0$.

3 Probabilistic Programs

In order to speak about probabilistic programs and the computations performed by such programs, we briefly introduce the syntax and semantics we use:

Definition 3 (Syntax). *Let Var be the set of program variables. The **set Prog of probabilistic programs** adheres to the following grammar:*

$$\text{Prog} \longrightarrow v := e \mid \text{Prog}; \text{Prog} \mid \{\text{Prog}\} [p] \{\text{Prog}\} \mid \text{WHILE } (b) \{\text{Prog}\},$$

where $v \in \text{Var}$, e is an arithmetical expression over Var , $p \in [0, 1] \subseteq \mathbb{Q}$, and b is a Boolean expression over arithmetic expressions over Var . We call the set of programs that do not contain any probabilistic choices the **set of ordinary programs** and denote this set by **ordProg**.

The presented syntax is the one of the fully probabilistic¹ fragment of the probabilistic guarded command language (pGCL) originally due to McIver and Morgan [4]. We omitted `skip`-, `abort`-, and `if`-statements, as those are syntactic sugar. While assignment, concatenation, and the while-loop are standard programming constructs, $\{P_1\} [p] \{P_2\}$ denotes a probabilistic choice between programs P_1 (with probability p) and P_2 (with probability $1 - p$). An operational semantics for pGCL programs is given below:

¹ Fully probabilistic programs may contain probabilistic but no non-deterministic choices.

Definition 4 (Semantics). Let the set of variable valuations be denoted by $\mathbb{V} = \{\eta \mid \eta: \text{Var} \rightarrow \mathbb{Q}^+\}$, let the set of program states be denoted by $\mathbb{S} = (\text{Prog} \cup \{\downarrow\}) \times \mathbb{V} \times I \times \{L, R\}^*$, for $I = [0, 1] \cap \mathbb{Q}^+$, let $\llbracket e \rrbracket_\eta$ be the evaluation of the arithmetical expression e in the variable valuation η , and analogously let $\llbracket b \rrbracket_\eta$ be the evaluation of the Boolean expression b . Then the **semantics of probabilistic programs** is given by the smallest relation $\vdash \subseteq \mathbb{S} \times \mathbb{S}$ which satisfies the following inference rules:

$$\begin{array}{c}
(\text{assign}) \frac{}{\langle v := e, \eta, a, \theta \rangle \vdash \langle \downarrow, \eta[v \mapsto \max\{\llbracket e \rrbracket_\eta, 0\}], a, \theta \rangle} \\
(\text{concat1}) \frac{\langle P_1, \eta, a, \theta \rangle \vdash \langle P'_1, \eta', a', \theta' \rangle}{\langle P_1; P_2, \eta, a, \theta \rangle \vdash \langle P'_1; P_2, \eta', a', \theta' \rangle} \\
(\text{concat2}) \frac{}{\langle \downarrow; P_2, \eta, a, \theta \rangle \vdash \langle P_2, \eta, a, \theta \rangle} \\
(\text{prob1}) \frac{}{\langle \{P_1\} [p] \{P_2\}, \eta, a, \theta \rangle \vdash \langle P_1, \eta, a \cdot p, \theta \cdot L \rangle} \\
(\text{prob2}) \frac{}{\langle \{P_1\} [p] \{P_2\}, \eta, a, \theta \rangle \vdash \langle P_2, \eta, a \cdot (1 - p), \theta \cdot R \rangle} \\
(\text{while1}) \frac{\llbracket b \rrbracket_\eta = \text{True}}{\langle \text{WHILE } (b) \{P\}, \eta, a, \theta \rangle \vdash \langle P; \text{WHILE } (b) \{P\}, \eta, a, \theta \rangle} \\
(\text{while2}) \frac{\llbracket b \rrbracket_\eta = \text{False}}{\langle \text{WHILE } (b) \{P\}, \eta, a, \theta \rangle \vdash \langle \downarrow, \eta, a, \theta \rangle}
\end{array}$$

We use $\sigma \vdash^k \tau$ in the usual sense.

The semantics is mostly straightforward except for two features: in addition to the program that is to be executed next and the current variable valuation, each state also stores a sequence θ that encodes which probabilistic choices were made in the past (Left or Right) as well as the probability a that those choices were made. The graph that is spanned by the \vdash -relation can be seen as an unfolding of the Markov decision process semantics for pGCL provided by Gretz *et al.* [5] when restricting oneself to fully probabilistic programs.

4 Expected Outcomes and Termination Probabilities

In this section we formally define the notion of an expected outcome as well the notion of (universal) (positive) almost-sure termination. We start by investigating how state successors can be computed.

It is a well-known result due to Kleene that for any ordinary program P and a state σ the k -th successor of σ with respect to \vdash is unique and computable. If, however, P is a probabilistic program containing probabilistic choices, the k -th successor of a state need not be unique, because at various points of the execution the program must choose a left or a right branch with some probability. However, if we resolve those choices by providing a sequence of symbols w over the alphabet

$\{L, R\}$ that encodes for all probabilistic choices which occur whether the *Left* or the *Right* branch shall be chosen at a branching point, we can construct a computable function that computes a unique k -th successor. Notice that for this purpose a sequence of finite length is sufficient. We obtain the following:

Proposition 1 (The State Successor Function). *Let $\mathbb{S}_\perp = \mathbb{S} \cup \{\perp\}$. There exists a total computable function $T: \mathbb{N} \times \mathbb{S} \times \{L, R\}^* \rightarrow \mathbb{S}_\perp$, such that for $k \geq 1$*

$$T_0(\sigma, w) = \begin{cases} \sigma, & \text{if } w = \varepsilon, \\ \perp, & \text{otherwise,} \end{cases}$$

$$T_k(\sigma, w) = \begin{cases} T_{k-1}(\tau, w'), & \text{if } \sigma = \langle P, \eta, a, \theta \rangle \vdash \langle P', \eta', a', \theta \cdot b \rangle = \tau, \\ & \text{with } w = b \cdot w' \text{ and } b \in \{L, R, \varepsilon\}, \\ \perp & \text{otherwise.} \end{cases}$$

So $T_k(\sigma, w)$ returns a successor state τ , if $\sigma \vdash^k \tau$, whereupon exactly $|w|$ inferences must use the (prob1)– or the (prob2)–rule and those probabilistic choices are resolved according to w . Otherwise $T_k(\sigma, w)$ returns \perp . Note in particular that for both the inference of a terminal state $\langle \downarrow, \eta, a, \theta \rangle$ within less than k steps as well as the inference of a terminal state through less or more than $|w|$ probabilistic choices, the calculation of $T_k(\sigma, w)$ will result in \perp . In addition to T , we will need two more computable operations for expressing expected outcomes, termination probabilities, and expected runtimes:

Proposition 2. *There exist two total computable functions $\alpha: \mathbb{S}_\perp \rightarrow \mathbb{Q}^+$ and $\wp: \mathbb{S}_\perp \times \text{Var} \rightarrow \mathbb{Q}^+$, such that*

$$\alpha(\sigma) = \begin{cases} a, & \text{if } \sigma = \langle \downarrow, _, a, _ \rangle \\ 0, & \text{otherwise,} \end{cases} \quad \wp(\sigma, v) = \begin{cases} \eta(v) \cdot a, & \text{if } \sigma = \langle \downarrow, \eta, a, _ \rangle \\ 0, & \text{otherwise,} \end{cases}$$

where $_$ represents an arbitrary value.

The function α takes a state σ and returns the probability of reaching σ . The function \wp takes a state σ and a variable v and returns the probability of reaching σ multiplied with the value of v in the state σ . Both functions do that only if the provided state σ is a terminal state. Otherwise they return 0. Based on the above notions, we now define expected outcomes, termination probabilities and expected times until termination:

Definition 5 (Expected Outcome, Termination Probability, and Expected Time until Termination). *Let $P \in \text{Prog}$, $\eta \in \mathbb{V}$, $v \in \text{Var}$, $\sigma_{P,\eta} = \langle P, \eta, 1, \varepsilon \rangle$, and for a finite alphabet A let $A^{\leq k} = \bigcup_{i=0}^k A^i$. Then*

1. the **expected outcome** of v after executing P on η , denoted $\mathbf{E}_{P,\eta}(v)$, is

$$\mathbf{E}_{P,\eta}(v) = \sum_{k=0}^{\infty} \sum_{w \in \{L, R\}^{\leq k}} \wp(T_k(\sigma_{P,\eta}, w), v),$$

2. the *probability that P terminates on η* , denoted $\mathbf{Pr}_{P,\eta}(\downarrow)$, is

$$\mathbf{Pr}_{P,\eta}(\downarrow) = \sum_{k=0}^{\infty} \sum_{w \in \{L, R\}^{\leq k}} \alpha(\mathbf{T}_k(\sigma_{P,\eta}, w)),$$

3. the *expected time until termination of P on η* , denoted $\mathbf{E}_{P,\eta}(\downarrow)$, is

$$\mathbf{E}_{P,\eta}(\downarrow) = \sum_{k=0}^{\infty} \left(1 - \sum_{w \in \{L, R\}^{\leq k}} \alpha(\mathbf{T}_k(\sigma_{P,\eta}, w)) \right).$$

The expected outcome $\mathbf{E}_{P,\eta}(v)$ as defined here coincides with the weakest pre-expectation $wp.P.v(\eta)$ à la McIver and Morgan [4] for fully probabilistic programs. In the above definition for $\mathbf{E}_{P,\eta}(v)$, we sum over all possible numbers of inference steps k and sum over all possible sequences from length 0 up to length k for resolving all probabilistic choices. Using \wp we filter out the terminal states σ and sum up the values of $\wp(\sigma, v)$.

For the termination probability $\mathbf{Pr}_P(\downarrow)$, we basically do the same but we merely sum up the probabilities of reaching final states by using α instead of \wp .

For the expected time until termination $\mathbf{E}_{P,\eta}(\downarrow)$, we go along the lines of [6]: It is stated there that the expected time until termination of P on η can be expressed as $\sum_{k=0}^{\infty} \Pr(\text{“}P \text{ runs for more than } k \text{ steps on } \eta\text{”}) = \sum_{k=0}^{\infty} (1 - \Pr(\text{“}P \text{ terminates within } k \text{ steps on } \eta\text{”}))$. We have expressed the latter in our set-up.

In order to investigate the complexity of calculating $\mathbf{E}_{P,\eta}(v)$, we define three sets: $\mathcal{L}\mathcal{E}\mathcal{X}\mathcal{P}$, which relates to the set of rational lower bounds of $\mathbf{E}_{P,\eta}(v)$, $\mathcal{R}\mathcal{E}\mathcal{X}\mathcal{P}$, which relates to the set of rational upper bounds, and $\mathcal{E}\mathcal{X}\mathcal{P}$ which relates to the value of $\mathbf{E}_{P,\eta}(v)$ itself:

Definition 6 ($\mathcal{L}\mathcal{E}\mathcal{X}\mathcal{P}$, $\mathcal{R}\mathcal{E}\mathcal{X}\mathcal{P}$, and $\mathcal{E}\mathcal{X}\mathcal{P}$). *The sets $\mathcal{L}\mathcal{E}\mathcal{X}\mathcal{P}$, $\mathcal{R}\mathcal{E}\mathcal{X}\mathcal{P}$, $\mathcal{E}\mathcal{X}\mathcal{P} \subseteq \text{Prog} \times \mathbb{V} \times \text{Var} \times \mathbb{Q}^+$ are defined as $(P, \eta, v, q) \in \mathcal{L}\mathcal{E}\mathcal{X}\mathcal{P}$ iff $q < \mathbf{E}_{P,\eta}(v)$, $(P, \eta, v, q) \in \mathcal{R}\mathcal{E}\mathcal{X}\mathcal{P}$ iff $q > \mathbf{E}_{P,\eta}(v)$, and $(P, \eta, v, q) \in \mathcal{E}\mathcal{X}\mathcal{P}$ iff $q = \mathbf{E}_{P,\eta}(v)$.*

Regarding the termination probability of a probabilistic program, the case of almost-sure termination is of special interest: We say that a program P *terminates almost-surely* on input η iff P terminates on η with probability 1. Furthermore, we say that P *terminates positively almost-surely* on η iff the expected time until termination of P on η is finite. Lastly, we say that P *terminates universally (positively) almost-surely*, if it does so on all possible inputs η . The problem of (universal) almost-sure termination can be seen as the probabilistic counterpart to the (universal) halting problem for ordinary programs.

In the following, we formally define the according problem sets:

Definition 7 (Almost-Sure Termination Problem Sets). *The sets $\mathcal{A}\mathcal{S}\mathcal{T}$, $\mathcal{P}\mathcal{A}\mathcal{S}\mathcal{T}$, $\mathcal{U}\mathcal{A}\mathcal{S}\mathcal{T}$, and $\mathcal{P}\mathcal{A}\mathcal{S}\mathcal{T}$ are defined as follows:*

$$\begin{aligned}
 (P, \eta) \in \mathcal{AST} &\iff \Pr_{P, \eta}(\downarrow) = 1 & (P, \eta) \in \mathcal{PAST} &\iff \mathbb{E}_{P, \eta}(\downarrow) < \infty \\
 P \in \mathcal{UAST} &\iff \forall \eta: (P, \eta) \in \mathcal{AST} & P \in \mathcal{UPAST} &\iff \forall \eta: (P, \eta) \in \mathcal{PAST}
 \end{aligned}$$

Notice that both $\mathcal{PAST} \subset \mathcal{AST}$ and $\mathcal{UPAST} \subset \mathcal{UAST}$ hold.

5 The Hardness of Computing Expected Outcomes

In this section we investigate the computational hardness of deciding the sets \mathcal{LEXP} , \mathcal{REXP} , and \mathcal{EXP} . The first fact we establish is the Σ_1^0 -completeness of \mathcal{LEXP} . This result is established by reduction from the (non-universal) halting problem for ordinary programs:

Theorem 1 (The Halting Problem [16]). *The halting problem is a subset $\mathcal{H} \subset \text{ordProg} \times \mathbb{V}$, which is characterized as $(P, \eta) \in \mathcal{H}$ iff $\exists k \exists \eta' : \mathbb{T}_k(\sigma_{P, \eta}, \varepsilon) = \langle \downarrow, \eta', 1, \varepsilon \rangle$. Let $\overline{\mathcal{H}}$ denote the complement of the halting problem, i.e. $\overline{\mathcal{H}} = (\text{ordProg} \times \mathbb{V}) \setminus \mathcal{H}$. \mathcal{H} is Σ_1^0 -complete and $\overline{\mathcal{H}}$ is Π_1^0 -complete.*

Theorem 2. *\mathcal{LEXP} is Σ_1^0 -complete.*

Proof. For showing $\mathcal{LEXP} \in \Sigma_1^0$, observe that $(P, \eta, v, q) \in \mathcal{LEXP}$ iff $\exists y: q < \sum_{k=0}^y \sum_{w \in \{L, R\}^{\leq k}} \wp(\mathbb{T}_k(\sigma_{P, \eta}, w), v)$, which is a Σ_1^0 -formula. Figure 1 (left) gives an intuition on this formula. For establishing Σ_1^0 -hardness we use a reduction function $f: \mathcal{H} \leq_m \mathcal{LEXP}$ with $f(Q, \eta) = (P, \eta, v, 1/2)$, where P is the program $v := 0; \{v := 1\}^{[1/2]}\{TQ; v := 1\}$ and TQ is an ordinary program that simulates Q on η . For details see [17]. \square

Theorem 2 implies that \mathcal{LEXP} is recursively enumerable. This means that all lower bounds for expected outcomes can be effectively enumerated by some algorithm. Now, if upper bounds were recursively enumerable as well, then expected outcomes would be computable reals. However, the contrary will be shown by establishing that \mathcal{REXP} is Σ_2^0 -complete, thus $\mathcal{REXP} \notin \Sigma_1^0$ and hence \mathcal{REXP} is not recursively enumerable. Σ_2^0 -hardness will be established by a reduction from the complement of the universal halting problem for ordinary programs:

Theorem 3 (The Universal Halting Problem [16]). *The universal halting problem is a subset $\mathcal{UH} \subset \text{ordProg}$, which is characterized as $P \in \mathcal{UH}$ iff $\forall \eta: (P, \eta) \in \mathcal{H}$. Let $\overline{\mathcal{UH}}$ denote the complement of \mathcal{UH} , i.e., $\overline{\mathcal{UH}} = \text{ordProg} \setminus \mathcal{UH}$. \mathcal{UH} is Π_2^0 -complete and $\overline{\mathcal{UH}}$ is Σ_2^0 -complete.*

Theorem 4. *\mathcal{REXP} is Σ_2^0 -complete.*

Proof. For showing $\mathcal{REXP} \in \Sigma_2^0$, observe that $(P, \eta, v, q) \in \mathcal{REXP}$ iff $\exists \delta \forall y: q - \delta > \sum_{k=0}^y \sum_{w \in \{L, R\}^{\leq k}} \wp(\mathbb{T}_k(\sigma_{P, \eta}, w), v)$, which is a Σ_2^0 -formula. Figure 1 (right) gives an intuition on this formula. For establishing Σ_2^0 -hardness we use a reduction function $f: \overline{\mathcal{UH}} \leq_m \mathcal{REXP}$ with $f(Q) = (P, \eta, v, 1)$, where η is arbitrary but fixed and P is the probabilistic program

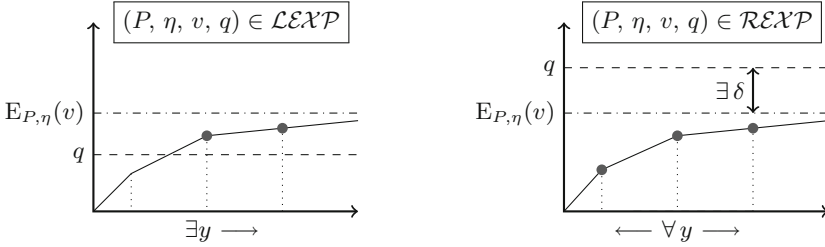


Fig. 1. Schematic depiction of the formulae defining \mathcal{LEXP} and \mathcal{REXP} , respectively. In each diagram, the solid line represents the monotonically increasing graph of $\sum_{k=0}^y \sum_{w \in \{L, R\} \leq k} \wp(\Gamma_k(\sigma_{P,\eta}, w), v)$ plotted over increasing y .

```

i := 0; {c := 0} [0.5] {c := 1};
while (c ≠ 0){i := i + 1; {c := 0} [0.5] {c := 1}};
k := 0; {c := 0} [0.5] {c := 1};
while (c ≠ 0){k := k + 1; {c := 0} [0.5] {c := 1}};
v := 0; TQ,
    
```

where TQ is a program that assigns the value 2^{k+1} to the variable v if and only if Q halts on input $g_Q(i)$ after exactly k steps (otherwise it assigns 0 to v) and $g_Q: \mathbb{N} \rightarrow \mathbb{V}$ is a computable bijection, such that $\forall z \in \text{Var}: (g_Q(i))(z) \neq 0$ implies that z occurs in Q . For details see [17]. \square

Finally, we establish the following result regarding exact expected outcomes:

Theorem 5. \mathcal{EXP} is Π_2^0 -complete.

Proof. For \mathcal{EXP} we can construct a Π_2^0 -formula from the two formulae defining \mathcal{LEXP} and \mathcal{REXP} . For establishing Π_2^0 -hardness we use the same reduction function f from the proof of Theorem 4 since for that function it holds that $f: \mathcal{UH} \leq_m \mathcal{EXP}$. For details see [17]. \square

6 The Hardness of Deciding Probabilistic Termination

This section presents the main contributions of this paper: Hardness results on several variations of almost-sure termination problems. We first establish that deciding almost-sure termination of a program on a given input is Π_2^0 -complete:

Theorem 6. \mathcal{AST} is Π_2^0 -complete.

Proof. For proving $\mathcal{AST} \in \Pi_2^0$, we show $\mathcal{AST} \leq_m \mathcal{EXP}$ using the reduction function $f: \mathcal{AST} \leq_m \mathcal{EXP}$ with $f(Q, \eta) = (P, \eta, v, 1)$, where v does not occur in Q and P is the program $v := 0; Q; v := 1$.

For establishing Π_2^0 -hardness we use a reduction function $f': \mathcal{UH} \leq_m \mathcal{AST}$ with $f'(Q) = (P', \eta)$, where η is arbitrary but fixed and P' is the program

```

i := 0; {c := 0} [0.5] {c := 1};
while (c ≠ 0){i := i + 1; {c := 0} [0.5] {c := 1}};
SQ,
    
```

where SQ is an ordinary program that simulates Q on $g_Q(i)$ and $g_Q: \mathbb{N} \rightarrow \mathbb{V}$ is the bijection from the proof of Theorem 4. For details see [17]. \square

While for ordinary programs there is a complexity leap when moving from the halting problem for some given input to the universal halting problem, we establish that *there is no such leap in the probabilistic setting*, i.e. \mathcal{UAST} is as hard as \mathcal{AST} :

Theorem 7. \mathcal{UAST} is Π_2^0 -complete.

Proof. For showing $\mathcal{UAST} \in \Pi_2^0$, consider that by Theorem 6 there must exist a decidable relation \mathcal{R} such that $(P, \eta) \in \mathcal{AST}$ iff $\forall y_1 \exists y_2: (y_1, y_2, P, \eta) \in \mathcal{R}$. By that we have that $P \in \mathcal{UAST}$ iff $\forall \eta \forall y_1 \exists y_2: (y_1, y_2, P, \eta) \in \mathcal{R}$, which is a Π_2^0 -formula.

It remains to show that \mathcal{UAST} is Π_2^0 -hard. This can be done by proving $\mathcal{AST} \leq_m \mathcal{UAST}$ as follows: On input (Q, η) the reduction function $f: \mathcal{AST} \leq_m \mathcal{UAST}$ computes a probabilistic program P that first initializes all variables according to η and then executes Q . \square

We now investigate the computational hardness of deciding *positive* almost-sure termination: It turns out that deciding \mathcal{PAST} is Σ_2^0 -complete. Thus, \mathcal{PAST} becomes semi-decidable when given access to an \mathcal{H} -oracle whereas \mathcal{AST} does not. We establish Σ_2^0 -hardness by a reduction from $\overline{\mathcal{UH}}$. This result is particularly counterintuitive as it means that for each ordinary program that *does not halt* on all inputs, we can *compute* a probabilistic program that *does halt* within an expected finite number of steps.

Theorem 8. \mathcal{PAST} is Σ_2^0 -complete.

Proof. For showing $\mathcal{PAST} \in \Sigma_2^0$, observe that $(P, \eta) \in \mathcal{PAST}$ iff $\exists c \forall \ell: c > \sum_{k=0}^{\ell} \left(1 - \sum_{w \in \{L, R\}^{\leq k}} \alpha(\mathbb{T}_k(\sigma_{P, \eta}, w))\right)$, which is a Σ_2^0 -formula. For more details on this formula see [17].

It remains to show that \mathcal{PAST} is Σ_2^0 -hard. For that we use a reduction function $f: \overline{\mathcal{UH}} \leq_m \mathcal{PAST}$ with $f(Q) = (P, \eta)$, where η is arbitrary but fixed and P is the program

```

c := 1; i := 0; x := 0; term := 0; InitQ;
while (c ≠ 0){
    StepQ; if (term = 1){Cheer; i := i + 1; term := 0; InitQ}
    {c := 0} [0.5] {c := 1}; x := x + 1 } ,
    
```

where $InitQ \in \text{ordProg}$ is a program that initializes a simulation of the program Q on input $g_Q(i)$ (recall the bijection $g_Q: \mathbb{N} \rightarrow \mathbb{V}$ from Theorem 4), $StepQ \in \text{ordProg}$ is a program that does one single (further) step of that simulation and sets **term** to 1 if that step has led to termination of Q , and $Cheer \in \text{ordProg}$ is a

program that executes 2^x many effectless steps. In the following we refer to this as “cheering”².

Correctness of the reduction: Intuitively, the program P starts by simulating Q on input $g_Q(0)$. During the simulation, it—figuratively speaking—gradually loses interest in further simulating Q by tossing a coin after each simulation step to decide whether to continue the simulation or not. If eventually P finds that Q has halted on input $g_Q(0)$, it “cheers” for a number of steps exponential in the number of coin tosses that were made so far, namely for 2^x steps. P then continues with the same procedure for the next input $g_Q(1)$, and so on.

The variable x keeps track of the number of loop iterations (starting from 0), which equals the number of coin tosses. The x -th loop iteration takes place with probability $1/2^x$. One loop iteration consists of a constant number of steps c_1 in case Q did not halt on input $g_Q(i)$ in the current simulation step. Such an iteration therefore contributes $c_1/2^x$ to the expected runtime of the probabilistic program P . In case Q did halt, a loop iteration takes a constant number of steps c_2 plus 2^x additional “cheering” steps. Such an iteration therefore contributes $c_2 + 2^x/2^x = c_2/2^x + 1 > 1$ to the expected runtime. Overall, the expected runtime of the program P roughly resembles a geometric series with exponentially decreasing summands. However, for each time the program Q halts on an input, a summand of the form $c_2/2^x + 1$ appears in this series. There are now two cases:

- (1) $Q \in \overline{UH}$, so there exists some input η with minimal i such that $g_Q(i) = \eta$ on which Q does not terminate. In that case, summands of the form $c_2/2^x + 1$ appear only $i - 1$ times in the series and therefore, the series converges—the expected time until termination is finite, so $(P, \eta) \in \mathcal{PAST}$.
- (2) $Q \notin \overline{UH}$, so Q terminates on every input. In that case, summands of the form $c_2/2^x + 1$ appear infinitely often in the series and therefore, the series diverges—the expected time until termination is infinite, so $(P, \eta) \notin \mathcal{PAST}$. □

The final problem we study is *universal* positive almost-sure termination. In contrast to the non-positive version, we do have a complexity leap when moving from non-universal to universal positive almost-sure termination. We will establish that $UPAST$ is Π_3^0 -complete and thus even harder to decide than $UAST$. For the reduction, we make use of the following Π_3^0 -complete problem:

Theorem 9 (The Cofiniteness Problem [16]). *The cofiniteness problem is a subset $\mathcal{COF} \subset \text{ordProg}$, which is characterized as $P \in \mathcal{COF}$ iff $\{\eta \mid (P, \eta) \in \mathcal{H}\}$ is cofinite. Let $\overline{\mathcal{COF}}$ denote the complement of \mathcal{COF} , i.e. $\overline{\mathcal{COF}} = \text{ordProg} \setminus \mathcal{COF}$. \mathcal{COF} is Σ_3^0 -complete and $\overline{\mathcal{COF}}$ is Π_3^0 -complete.*

Theorem 10. *$UPAST$ is Π_3^0 -complete.*

Proof. By Theorem 8, there exists a decidable relation \mathcal{R} such that $(P, \eta) \in \mathcal{PAST}$ iff $\exists y_1 \forall y_2: (y_1, y_2, P, \eta) \in \mathcal{R}$. Therefore $UPAST$ is definable by $P \in UPAST$ iff $\forall \eta \exists y_1 \forall y_2: (y_1, y_2, P, \eta) \in \mathcal{R}$, which gives a Π_3^0 -formula.

² The program P cheers as it was able to prove the termination of Q on input $g_Q(i)$.

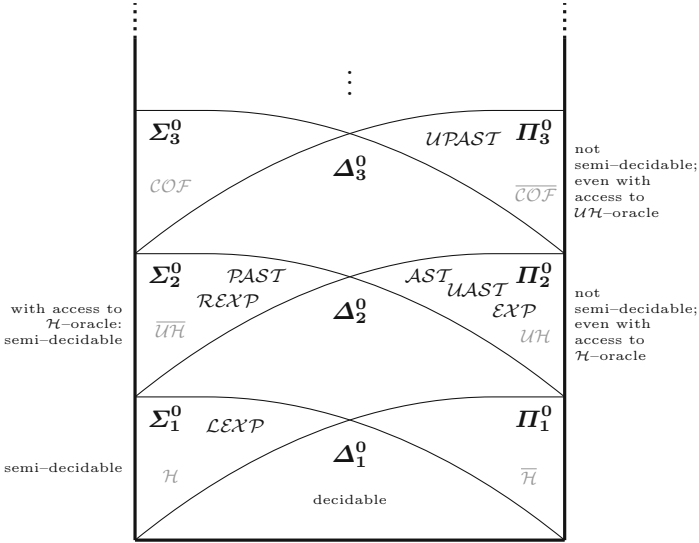


Fig. 2. The complexity landscape of determining expected outcomes and deciding (universal) (positive) almost-sure termination.

It remains to show that UPAST is Π_3^0 -hard. This is established by a reduction function $f: \text{COF} \leq_m \text{UPAST}$ such that $f(Q)$ gives nearly the same program as the reduction function from the proof of Theorem 8 except that in $f(Q)$ the initialization $i := 0$ is omitted. Thus, on input η the resulting program P also simulates Q successively on all inputs but starting from input $g_Q(\eta(i))$ instead of $g_Q(0)$. For details, see [17]. \square

7 Conclusion

We have studied the computational complexity of solving a variety of natural problems which appear in the analysis of probabilistic programs: Computing lower bounds, upper bounds, and exact expected outcomes (LEXP , REXP , and EXP), deciding non-universal and universal almost-sure termination (AST and UAST), and deciding non-universal and universal positive almost-sure termination (PAST and UPAST). Our complexity results are summarized in Fig. 2. All examined problems are complete for their respective level of the arithmetical hierarchy. We conjecture that all our results remain valid for programs with (minimizing) demonic non-determinism à la McIver and Morgan [4].

Future work consists of identifying program subclasses for which some of the studied problems become easier. One idea towards this would be to investigate the use of quantifier-elimination methods such as e.g. Skolemization.

Acknowledgements. We would like to thank Luis María Ferrer Fioriti (Saarland University) and Federico Olmedo (RWTH Aachen) for the fruitful discussions on the topics of this paper. Furthermore, we are very grateful for the valuable and constructive comments we received from the anonymous referees on an earlier version of this paper.

References

1. Kozen, D.: Semantics of Probabilistic Programs. *J. Comput. Syst. Sci.* **22**(3), 328–350 (1981)
2. Barthe, G., Köpf, B., Olmedo, F., Béguelin, S.Z.: Probabilistic relational reasoning for differential privacy. *ACM Trans. Program. Lang. Syst.* **35**(3), 9 (2013)
3. Borgström, J., Gordon, A., Greenberg, M., Margetson, J., van Gael, J.: Measure transformer semantics for Bayesian machine learning. *LMCS* **9**(3), 1–39 (2013). Paper Number 11
4. McIver, A., Morgan, C.: *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, New York (2004)
5. Gretz, F., Katoen, J.P., McIver, A.: Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. *Perform. Eval.* **73**, 110–132 (2014)
6. Fioriti, L.M.F., Hermanns, H.: Probabilistic termination: Soundness, completeness, and compositionality. In: *POPL 2015*, pp. 489–501. ACM (2015)
7. Sneyers, J., De Schreye, D.: Probabilistic termination of CHRiSM programs. In: Vidal, G. (ed.) *LOPSTR 2011*. LNCS, vol. 7225, pp. 221–236. Springer, Heidelberg (2012)
8. Arons, T., Pnueli, A., Zuck, L.D.: Parameterized verification by probabilistic abstraction. In: Gordon, A.D. (ed.) *FOSSACS 2003*. LNCS, vol. 2620, pp. 87–102. Springer, Heidelberg (2003)
9. Esparza, J., Gaiser, A., Kiefer, S.: Proving termination of probabilistic programs using patterns. In: Madhusudan, P., Seshia, S.A. (eds.) *CAV 2012*. LNCS, vol. 7358, pp. 123–138. Springer, Heidelberg (2012)
10. Morgan, C.: Proof rules for probabilistic loops. In: *Proceedings of the BCS-FACS 7th Refinement Workshop, Workshops in Computing*. Springer (1996)
11. Tiomkin, M.L.: Probabilistic termination versus fair termination. *TCS* **66**(3), 333–340 (1989)
12. Kleene, S.C.: Recursive predicates and quantifiers. *Trans. AMS* **53**(1), 41–73 (1943)
13. Odifreddi, P.: *Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers*. Elsevier, Amsterdam (1992)
14. Post, E.L.: Recursively enumerable sets of positive integers and their decision problems. *Bull. AMS* **50**(5), 284–316 (1944)
15. Davis, M.D.: *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*. Academic Press, Cambridge (1994)
16. Odifreddi, P.: *Classical Recursion Theory*, vol. II. Elsevier, Amsterdam (1999)
17. Kaminski, B.L., Katoen, J.P.: On the Hardness of Almost-Sure Termination. *ArXiv e-prints*, June 2015

Graphs Identified by Logics with Counting

Sandra Kiefer^(✉), Pascal Schweitzer, and Erkal Selman

RWTH Aachen University, Aachen, Germany
{kiefer,schweitzer,selman}@informatik.rwth-aachen.de

Abstract. We classify graphs and, more generally, finite relational structures that are identified by C^2 , that is, two-variable first-order logic with counting. Using this classification, we show that it can be decided in almost linear time whether a structure is identified by C^2 . Our classification implies that for every graph identified by this logic, all vertex-colored versions of it are also identified. A similar statement is true for finite relational structures.

We provide constructions that solve the inversion problem for finite structures in linear time. This problem has previously been shown to be polynomial time solvable by Martin Otto. For graphs, we conclude that every C^2 -equivalence class contains a graph whose orbits are exactly the classes of the C^2 -partition of its vertex set and which has a single automorphism witnessing this fact.

For general k , we show that such statements are not true by providing examples of graphs of size linear in k which are identified by C^3 but for which the orbit partition is strictly finer than the C^k -partition. We also provide identified graphs which have vertex-colored versions that are not identified by C^k .

1 Introduction

The k -variable fragment of counting logic, denoted by C^k , is obtained from first-order logic by adding counting quantifiers but only allowing formulas that use at most k variables. These finite variable logics play a central role in the area of model-checking since for them the model-checking problem and the equivalence problem are solvable in polynomial time (see [11]). For a while, there was the hope that for some fixed k the logic C^k can distinguish every pair of non-isomorphic graphs. This would imply that the graph isomorphism problem is solvable in polynomial time. However, in 1992, it was shown by Cai, Fürer and Immerman [6] that $\Omega(n)$ variables are required to identify all graphs on n vertices. Since the examples presented in that paper consist of graph isomorphism instances which are actually known to be solvable in polynomial time, this also shows that C^k does not capture polynomial time.

Concerning C^k , there are striking connections to other seemingly unrelated areas. For example, there exist several Ehrenfeucht-Fraïssé type games characterizing C^k [6, 8, 13]. Also strongly related is the $(k - 1)$ -dimensional version of a well-known color refinement algorithm, named after Weisfeiler and Lehman by

Babai (see [6]). It turns out that the $(k-1)$ -dimensional Weisfeiler-Lehman algorithm does nothing else but partition $(k-1)$ -tuples of vertices according to their C^k -types. Another surprising connection exists to linear programming. The k -th level of the Sherali-Adams hierarchy of a natural linear integer programming formulation of graph isomorphism essentially corresponds to the expressive power of C^k [2, 12].

Among the finite variable logics, the fragment C^2 has been of particular interest because its satisfiability problem is known to be decidable [10], and the complexity of the decision problem has been studied extensively [22]. Numerous results for this logic are known, we refer the reader to a survey by Grädel and Otto [9]. In practice, due to its strength and the fact that it can be evaluated in almost linear time, the logic C^2 (more specifically, the corresponding 1-dimensional Weisfeiler-Lehman algorithm) is an essential subroutine in all competitive canonical labeling tools (see [20]). Very recent results concerning C^2 include a paper by Krebs and Verbitsky studying the quantifier depth of C^2 -formulas for C^2 -equivalence classes of graphs [17]. Kopczynski and Tan show that for every fixed C^2 -formula, the set of those n for which there is a structure with a universe of size n satisfying the formula is semilinear [16]. Moreover, they describe the characteristics of the spectrum of a C^2 -formula.

While most of the results further above deal with the problem of distinguishing two graphs from each other using finite variable counting logics, in this paper we are concerned with the concept of distinguishing a graph from every other non-isomorphic graph. We say that the graph is **identified** by the logic. More formally, a graph (or a finite relational structure) G is identified by a logic L if there is a sentence φ in L such that G satisfies φ and every graph (or finite relational structure) which satisfies φ is isomorphic to G .

Of course every graph is identified by some first-order sentence. However, by [6], as mentioned above, there is no $k \in \mathbb{N}$ such that every graph is identified by some formula in C^k . Let us focus on the case $k = 2$. It is not difficult to see that all forests are identified by C^2 . Moreover, a graph is asymptotically almost surely identified by C^2 , that is, the fraction of graphs of size n which are not identified by C^2 tends to 0 as n tends to infinity [3]. Even more strongly, it is known [4] that the fraction of graphs which are not identified is exponentially small in n . Similarly, a regular graph is asymptotically almost surely identified by C^3 [18]. However, not all graphs are identified. (For C^2 , consider a cycle of length at least 6, for example.) The following question arises.

What is the structure of graphs that are identified by C^k ?

Our results. We study graphs that are identified by C^2 and provide a complete classification for them. This classification can be used to draw several conclusions about general properties of identified graphs. For example, one can derive that if an undirected graph is identified by C^2 , then the C^2 -partition classes of the vertices are exactly the orbits of the automorphism group of the graph. This corollary is neither true when considering finite (relational) structures nor when considering C^k with $k > 2$. For C^2 , we also conclude that if an undirected graph is identified, every vertex-colored version of it is identified by C^2 as well.

This statement holds for finite relational structures, too, but is again not true for C^k with $k > 2$. Using our classification, we show that in time $O((n+m) \log n)$ it is possible to determine whether an undirected graph is identified by C^2 .

The proof of the correctness of our classification hinges upon explicit constructions that solve the inversion problem and the canonization problem for C^2 . The inversion problem asks whether to a certain invariant a graph (or more generally, a finite structure) can be constructed. In the case of C^2 , such an invariant is the count of the C^2 -types of pairs of vertices. A celebrated result by Otto [21] shows that the inversion problem and the canonization problem for C^2 can be solved in polynomial time. As a side-product, our direct constructions provide an alternative proof for this. In fact, we show that the inversion problem for C^2 can be solved in linear time. Our constructions make use of circulant graphs and doubly-circulant graphs. With these, we observe that every C^2 -equivalence class contains a graph whose C^2 -partition classes are the orbits. More strongly, there is a single automorphism of the graph witnessing this. (That is, there is an automorphism φ such that for all pairs of vertices v, v' that are in the same C^2 -partition class there is an integer i such that $\varphi^i(v) = v'$.) To achieve inversion for finite structures, we use an old 1-factorization construction due to Walecki (see [19]) that decomposes the complete graph K_{2n} into $2n - 1$ disjoint perfect matchings.

Building on the classification of graphs identified by C^2 , we also classify finite structures that are identified by C^2 . For graphs, there is only one special case that may appear within a C^2 -partition class (namely the cycle of length 5). However, for finite structures there are 7 different special cases for a C^2 -partition class, which are of sizes 3, 4, 5 and 6. Our classification theorem describes how these may be combined to form structures that are identified by C^2 . Due to the nature of the different special cases, the classification is more involved (see Theorem 4). Nevertheless, we show that one can decide in almost linear time whether a structure is identified by C^2 . Our characterization also provides a graph-theoretical classification result. From the class of (possibly edge-colored partially oriented) graphs, it explicitly lists all those (color-)regular graphs that are determined up to isomorphism by their (color) degrees, see Theorem 3.

For the logics C^k with $k > 2$ we collect several negative results. One can first observe that the triangular graphs form an infinite non-trivial class of strongly regular graphs which are identified by C^3 , implying that any classification result would have to include non-trivial infinite families [7, 14].

Contrasting our results for C^2 , we provide examples of graphs that are identified by C^3 but for which conclusions analogous to the ones mentioned above do not hold. More specifically, we present graphs identified by C^3 for which even the logic C^k with k linear in the size of the graph does not correctly determine the orbit partition. This yields graphs which the logic C^k identifies, but for which not all vertex-colored versions are identified by C^k . These ideas are based on the construction by Cai, Fürer and Immerman [6].

The existence of these graphs highlights an important fact. Even if a graph is identified by the logic C^k , it is not clear that it is possible to take advantage

of that in order to canonize the graph. This stands in contrast to a remark in [6] claiming that a graph G identified by C^k can be canonized in polynomial time. In fact, the crucial property required for the approach hinted at there to be successful is that all vertex-colored versions of G need to be identified by C^k . Indeed, if this property holds then a standard recursive individualization approach canonizes the graph G . It would suffice to show that for all vertex-colored versions of G the orbits are determined by the logic. However, with a slight alteration of our construction, we obtain a graph G that is identified by C^k and whose orbits are correctly determined, but for which there exist vertex-colored versions whose orbits are not correctly determined.

Independently of our work, Arvind, Köbler, Rattan and Verbitsky [1] have investigated the structure of undirected graphs identified by C^2 obtaining results similar to the ones we provide in Sect. 4. Throughout the document, proofs are omitted. For these, we refer to the full version [15].

2 Preliminaries

Unless specified otherwise, a **graph** G is a finite undirected graph without loops, its vertex set is $V(G)$ and its edge set $E(G)$. For $P \subseteq V(G)$, the subgraph induced by P is $G[P]$. If all vertices have degree k then G is **k -regular**. A **(k, ℓ) -biregular graph G on bipartition (P, Q)** is a graph on vertex set $P \dot{\cup} Q$ such that P and Q are independent sets, every vertex in P has exactly k neighbors in Q and every vertex in Q has exactly ℓ neighbors in P . Two vertices $v, v' \in V(G)$ are in the same **orbit of G** if G has an automorphism φ such that $\varphi(v) = v'$. The **orbit partition** of G is the partition of $V(G)$ into the orbits of G .

A graph G is **identified** by a logic L if there is a sentence φ in L such that G satisfies φ and every graph which satisfies φ is isomorphic to G . A logic L **distinguishes** two graphs G and G' if there is a sentence φ in L such that $G \models \varphi$ and $G' \not\models \varphi$. We say that G and G' are **L -equivalent** if they are not distinguished by L .

The **k -variable counting logic**, denoted by C^k , is the k -variable fragment of first-order logic enriched by counting quantifiers. For every $t \in \mathbb{N}$ we have the counting quantifier $\exists^{\geq t}$. For a formula $\varphi(x)$ with the free variable x and for a graph G we have $G \models \exists^{\geq t} x \varphi(x)$ if and only if there are at least t vertices $v \in V(G)$ such that $G \models \varphi[v]$. The variables in a C^k -formula are all from a fixed k -element set, say $\{x_1, \dots, x_k\}$, but they can be reused. The **C^k -type** of a vertex v in G is the set of all C^k -formulas $\varphi(x)$ such that $G \models \varphi[v]$. The **C^k -coloring** of G is the coloring of each vertex with its C^k -type. The **C^k -partition** of a graph G is the partition of its vertex set induced by their C^k -types. Similarly, the **C^k -type** of a tuple (v, w) is the set of C^k -formulas $\varphi(x, y)$ such that $G \models \varphi[v, w]$. For a vertex v or a pair of vertices v, w to obtain the **atomic C^k -type** we consider only quantifier-free C^k -formulas.

Let G be a graph and Π be a partition of $V(G)$. We say that Π is **equitable** if for all $P, Q \in \Pi$ and $v, v' \in P$, the vertices v and v' have the same number of neighbors in Q . A vertex coloring χ is called equitable if the partition induced

by χ is equitable. A partition Π is **coarser** than a partition Π' if every partition class of Π is contained in some class of Π' . It is a well-known fact that the C^2 -partition of a graph is its **coarsest equitable partition**. The C^2 -partition of a graph with n vertices and m edges can be calculated in time $O((m+n)\log n)$ by the color refinement procedure (see [5]).

2.1 Relational Structures and Partially Oriented Graphs

An **edge-colored partially oriented graph** (an ec-POG) is an edge-colored directed graph (G, c) (with c an edge-coloring function) without loops such that for every $(v, w) \in E(G)$, it holds that if $(w, v) \in E(G)$ then $c((v, w)) = c((w, v))$. Slightly abusing terminology, we say that an edge $(v, w) \in E(G)$ is **undirected** if $(w, v) \in E(G)$ and **directed** otherwise. We accordingly draw (v, w) and (w, v) as one undirected edge between v and w and denote it by $\{v, w\}$. An ec-POG (G, c) is **complete** if for all $v, w \in V(G)$ with $v \neq w$ we have $(v, w) \in E(G)$ or $(w, v) \in E(G)$.

In the following, we consider finite relational structures over a fixed signature $\sigma = (R_1, \dots, R_\ell)$ where R_i has arity r_i . The various definitions given for graphs are analogously defined for structures. Let \mathfrak{A} be a finite relational structure with universe A . For every $i \in \{1, \dots, \ell\}$ we define a function $c_i : A^2 \rightarrow \mathcal{P}(\{1, 2\}^{r_i})$ via $c_i(v_1, v_2) := \{(j_1, \dots, j_{r_i}) \in \{1, 2\}^{r_i} \mid \mathfrak{A} \models R_i(v_{j_1}, \dots, v_{j_{r_i}})\}$ where, as usual, \mathcal{P} denotes the power set and $\{1, 2\}^{r_i}$ denotes the set of all r_i -tuples over $\{1, 2\}$. For all $v, w \in A$ with $v \neq w$ we let $c(v, w) := (c_1(v, w), \dots, c_\ell(v, w))$. Since for each i the possible images of c_i come from a set of bounded size, by using the order of the relations R_i in σ , one can easily define a canonical linear ordering \leq on the image of c (for example, by using the lexicographic order). With the help of this ordering, we define $\text{ec-POG}(\mathfrak{A}) := ((A, E_{\mathfrak{A}}), c_{\mathfrak{A}})$ as the complete ec-POG with vertex set A , edge set $E_{\mathfrak{A}} := \{(v, w) \mid v, w \in A, v \neq w \text{ and } c(v, w) \leq c(w, v)\}$ and the edge coloring $c_{\mathfrak{A}} := c|_{E_{\mathfrak{A}}}$, the restriction of c to $E_{\mathfrak{A}}$. Note that $c(v, w)$ uniquely determines the atomic C^2 -types of $v, w, (v, w)$ and (w, v) .

A partition Π of the vertex set of an ec-POG is **equitable** if for all $P, Q \in \Pi$, for all $v, v' \in P$ and for every edge color c , the vertices v and v' have the same number of c -colored outgoing, incoming and undirected edges connecting them to Q . An ec-POG is **color-regular** if for each edge color c every vertex v has the same c -indegree, the same c -outdegree and the same c -degree for undirected edges, respectively. An edge-colored undirected biregular graph on bipartition (P, Q) is called **color-biregular** if for every edge color c the subgraph induced by the edges of color c is biregular on (P, Q) . If the graph is partially oriented, vertices in each bipartition class must additionally have the same number of outgoing and incoming edges in each color.

3 Inversion

In this section, we treat the so-called **inversion problem** that is closely related to the question which graphs are identified by C^2 . A **complete invariant** of an

equivalence relation \equiv on a class \mathcal{C} of structures is a mapping \mathcal{I} from \mathcal{C} to some set S , such that $\mathfrak{A} \equiv \mathfrak{B}$ if and only if $\mathcal{I}(\mathfrak{A}) = \mathcal{I}(\mathfrak{B})$. We say that \mathcal{I} **admits linear time inversion** if given $s \in S$ one can construct in linear time a structure \mathfrak{A} with $\mathcal{I}(\mathfrak{A}) = s$ or decide that no such structure exists. This algorithmic task describes the inversion problem. Of course, if a structure \mathfrak{A} is identified then a solution to the inversion problem must construct \mathfrak{A} when given $\mathcal{I}(\mathfrak{A})$.

For C^2 , we show that a natural complete invariant, namely \mathcal{I}_C^2 , admits linear time inversion. Otto [21] proved that this invariant admits polynomial time inversion, not only for simple graphs, but for finite structures in general.

Given a graph G , one can canonically define a linear ordering $P_1 \leq \dots \leq P_t$ on the classes of its coarsest equitable partition (see [21]). This ordering allows us to define \mathcal{I}_C^2 , mapping G to (\bar{s}, M) , where \bar{s} is the tuple $(|P_1|, \dots, |P_t|)$ and M is a $t \times t$ matrix, such that every vertex in P_i has exactly M_{ij} neighbors in P_j . It is easy to see that \mathcal{I}_C^2 is a complete invariant of C^2 .

3.1 Inversion for Graphs

Definition 1. A graph is **circulant** if it has an automorphism that consists of exactly one (permutation) cycle. A graph on vertex set $P \cup Q$ is **doubly-circulant** with respect to P and Q if it has an automorphism with exactly two (permutation) cycles, one on P and the other on Q . A graph is **multi-circulant** with respect to a partition $\{P_1, \dots, P_\ell\}$ of its vertices if it has an automorphism with exactly ℓ (permutation) cycles, each on one of the P_i .

While circulant graphs are transitive and thus regular, not every regular graph is transitive and not every transitive graph is circulant. Similarly, every doubly-circulant graph is biregular but not every biregular graph is doubly-circulant.

It is well-known that circulant graphs can be constructed by numbering the vertices from 0 to $n - 1$, picking an arbitrary set $S \subseteq \{1, \dots, \lfloor n/2 \rfloor\}$ of distances and inserting all edges between pairs of vertices whose distance of indices in the circular ordering is contained in S . A k -regular graph on n vertices exists exactly if $k \cdot n$ is even and $k \leq n - 1$. In this case a k -regular circulant graph on n vertices can be constructed in linear time. We call this the **circulant construction**.

A (k, ℓ) -biregular graph on a bipartition (P, Q) with $|P| = m$ and $|Q| = n$ exists exactly if $k \cdot m = \ell \cdot n$ as well as $k \leq n$ and $\ell \leq m$ (see, for example [16, 21]). In fact, under these conditions there exists such a graph that is doubly-circulant.

Lemma 1. For all $k, \ell, m, n \in \mathbb{N}$ with $k \leq n, \ell \leq m$ and $k \cdot m = \ell \cdot n$, one can construct in $O(k \cdot m)$ time a (k, ℓ) -biregular graph on a bipartition (P, Q) with $|P| = m$ and $|Q| = n$, which is doubly-circulant with respect to P and Q .

The coarsest equitable partition of a graph is not necessarily its orbit partition. However, one can use Lemma 1 to show that for each C^2 -equivalence class, there is a representative whose coarsest equitable partition is the orbit partition.

Theorem 1. For every graph G , there is a C^2 -equivalent graph H which is multi-circulant with respect to its coarsest equitable partition.

Corollary 1. I_C^2 admits linear time inversion on the class of graphs.

Given an equivalence relation \equiv on a class \mathcal{C} of structures, the **canonization problem** for \equiv is the problem of finding a map $c : \mathcal{C} \rightarrow \mathcal{C}$ such that for every $\mathfrak{A} \in \mathcal{C}$ it holds that $c(\mathfrak{A}) \equiv \mathfrak{A}$ and for all $\mathfrak{A}, \mathfrak{B} \in \mathcal{C}$ with $\mathfrak{A} \equiv \mathfrak{B}$ we have $c(\mathfrak{A}) = c(\mathfrak{B})$. The map c is called a **canonization** for \equiv and $c(\mathfrak{A})$ is the **canon** of \mathfrak{A} (with respect to c). Typically, the goal is to find such a canonization c that can be evaluated efficiently. As a consequence of the theorem we obtain two corollaries.

Corollary 2. Canonization for C^2 of graphs can be done in $O((n + m) \log n)$ time.

Corollary 3. If a graph G is identified by C^2 , then its coarsest equitable partition is the orbit partition.

It is natural to ask whether Corollary 3 holds for finite relational structures in general. This is not the case since the unique 1-factorization of K_6 is rigid (i.e., has no non-trivial automorphisms).

3.2 Inversion for Finite Relational Structures

Let $\mathfrak{A} = (A, R_1, \dots, R_\ell)$ be a finite relational structure. We define $\mathfrak{A}|_2$, the restriction of \mathfrak{A} to arity 2, to be the relational structure $(A, R'_1, \dots, R'_\ell)$ with $R'_i := \{(v_1, \dots, v_{r_i}) \in R_i \mid \{v_1, \dots, v_{r_i}\} \text{ has at most 2 elements,}\}$ where r_i is the arity of R_i . Obviously, two relational structures \mathfrak{A} and \mathfrak{B} are C^2 -equivalent if and only if $\mathfrak{A}|_2$ and $\mathfrak{B}|_2$ are C^2 -equivalent. Furthermore, $\mathfrak{A}|_2$ and $\mathfrak{B}|_2$ are C^2 -equivalent if and only if $\text{ec-POG}(\mathfrak{A}|_2)$ and $\text{ec-POG}(\mathfrak{B}|_2)$ are C^2 -equivalent. Hence, the inversion problem for I_C^2 on finite relational structures reduces to the inversion problem for I_C^2 on ec-POGs.

The complete invariant I_C^2 for the class of graphs has a well-known extension to finite structures, which translates to the context of ec-POGs as follows: we simply replace the matrix in the original definition of I_C^2 with a matrix M such that for all $i, j \in \{1, \dots, k\}$ the entry M_{ij} is a tuple encoding for each edge color d the number of d -colored outgoing edges from a vertex in P_i to P_j . Similarly to the case of graphs, in order to solve the inversion problem for ec-POGs it suffices to solve it for the color-regular case (which corresponds to the inversion within one C^2 -partition class) and for the color-biregular case (which corresponds to the inversion between two C^2 -partition classes).

Color-regular case: Let n be the number of vertices. If n is odd, the degrees in each color in the underlying undirected graph must be even. Consequently, the circulant construction from Subsect. 3.1 can be adapted to perform the inversion for directed and colored edges.

If n is even and more than one color degree is odd, we cannot apply the circulant construction, which significantly complicates our task. In fact, as a 1-factorization (i.e., a partition of the edge set into perfect matchings) of K_6 shows it might not be possible to construct a transitive graph with the given

color degrees. Still, we can construct a canonical representative for the corresponding C^2 -equivalence class as follows. We use the 1-factorization construction due to Walecki (see [19]). For even n the construction decomposes the complete graph K_n into $n - 1$ disjoint perfect matchings. This decomposition can be made canonical and computed in linear time (see also [23, Example 7.1.2.]). To obtain a graph with specific color degrees we can define a color class to be the union of a suitable number of 1-factors. We call this the **matching construction**. The construction has the property that two matchings of the 1-factorization always yield a Hamiltonian cycle. If we require directed edges, in which case in-degrees must be equal to out-degrees, we pair a suitable number of matchings and orient the obtained Hamiltonian cycles.

Color-biregular case: Our doubly-circulant construction for graphs can be altered to handle colored directed edges.

Corollary 4. \mathcal{I}_C^2 admits linear time inversion on finite relational structures.

Corollary 5. Canonization for C^2 of a relational structure $\mathfrak{A} = (A, R_1, \dots, R_\ell)$ over a fixed signature can be done in time $O((n + m) \log n)$ where $n = |A|$ and $m = |R_1| + \dots + |R_\ell|$.

4 Characterization of the Graphs Identified by C^2

Here we examine the graphs that are identified by the logic C^2 and give a complete characterization of them.

Definition 2. Let T be a tree with a designated vertex v . For $i \in \{1, \dots, 5\}$ let (T_i, v_i) be an isomorphic copy of (T, v) . Let F be the disjoint union of the five trees (T_i, v_i) and E be the edge set of a 5-cycle on vertex set $\{v_1, \dots, v_5\}$. Then we call the graph obtained from F by inserting the edges in E , a **bouquet**.

A **bouquet forest** is a disjoint union of vertex-colored trees and non-isomorphic vertex-colored bouquets.

For sets P, Q we define $[P, Q] := \{\{v, w\} \mid v \neq w, v \in P, w \in Q\}$.

Definition 3. Let G be a graph with C^2 -coloring χ . We define the **flip of G** as the vertex-colored graph (F, χ) with $V(F) = V(G)$ and $E(F) = E(G) \Delta ([P_1, Q_1] \cup \dots \cup [P_t, Q_t])$, where the (P_i, Q_i) are all pairs of (not necessarily distinct) C^2 -partition classes of G which satisfy $|[P_i, Q_i] \cap E(G)| > |[P_i, Q_i] \setminus E(G)|$. We say that (F, χ) is a **flipped graph**. If χ is the C^2 -coloring of F and (F, χ) is a flipped graph, we also say that F is flipped.

Here, the symbol Δ in the definition denotes, the symmetric difference. The notions of a flip and a bouquet forest allow us to obtain the following classification.

Theorem 2. A graph is identified by C^2 if and only if its flip is a bouquet forest.

Corollary 6. Given a graph with n vertices and m edges, we can decide whether it is identified by C^2 in time $O((m + n) \log n)$.

A second corollary of Theorem 2 is concerned with vertex colorings of graphs that are identified by C^2 .

Corollary 7. *Let (G, χ) be a vertex-colored graph which is identified by C^2 and let χ' be a vertex coloring of G which induces a finer partition on $V(G)$ than χ does. Then (G, χ') is also identified by C^2 .*

Our classification result actually provides us with some deeper structural insight that we describe next.

For a (k, ℓ) -biregular graph on bipartition (P, Q) we introduce the three following notations. (1) $P \square Q \iff k = \ell = 0$, (2) $P \doteq Q \iff k = \ell = 1$ and (3) $P \ll Q \iff k \geq 2$ and $\ell = 1$.

For a graph G with C^2 -partition Π , we define the **skeleton** S_G of G as the graph with $V(S_G) = \Pi$ and $E(S_G) = \{\{P, Q\} \mid P \doteq Q \text{ or } P \ll Q \text{ in the flip of } G\}$.

Since they can appear only once per connected component of the skeleton, we call a C^2 -partition class that is a 5-cycle or a matching an **exception**. Our classification of finite relational structures that are identified by C^2 , which we give in the next section, depends on the structural properties that can be proven for identified graphs. These can be summarized as follows.

Corollary 8. *A flipped graph G is identified by C^2 if and only if the following hold: (1) Each C^2 -partition class induces a graph identified by C^2 (i.e., the induced graph has no edges or it is a matching or a 5-cycle), (2) for all C^2 -partition classes P and Q we have $P \square Q$, $P \doteq Q$, $P \ll Q$ or $Q \ll P$, (3) the skeleton S_G is a forest, (4) there is no path P_0, P_1, \dots, P_t in S_G with $P_0 \ll P_1$ and $P_{t-1} \gg P_t$ in G , (5) there is no path P_0, P_1, \dots, P_t in S_G where $P_0 \ll P_1$ and $G[P_t]$ is a 5-cycle or a matching, and (6) in every connected component of S_G there is at most one exception.*

5 General Finite Structures

To generalize our results to finite structures, it suffices to analyze which edge-colored partially oriented graphs (i.e., ec-POGs) are identified.

Theorem 3. *Let G be a color-regular complete ec-POG. Then C^2 identifies G if and only if G is (1) an undirected complete graph with only one edge color, (2) undirected and has two edge colors, one of which induces a perfect matching, or (3) one of the exceptions depicted in Fig. 1.*

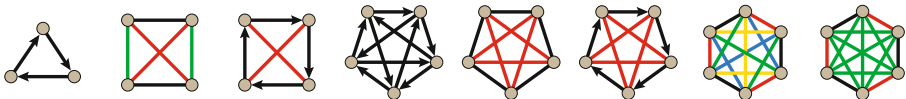


Fig. 1. The special cases that occur in the classification in Theorem 3

We will now describe how to combine such building blocks to form a larger identified ec-POG. By interpreting non-edges as edges of a special color we only need to consider complete graphs.

Definition 4. Let G be a vertex-colored ec-POG and let P and Q be two disjoint subsets of $V(G)$. We introduce the relation $P \equiv_3^3 Q$ to denote the fact that the graph induced by the edges running between P and Q is the graph $K_{3,3}$ with three edge colors which each induce a perfect matching between P and Q .

To define the relations $P \square Q, P \doteq Q, P \ll Q$ for edge-colored graphs we always consider the graph induced by the edge color class that contains fewer edges and ignore orientations. It is not difficult to see that if the number of edges in the first color is equal to the number of edges in the second color then choosing either induced graph yields the same results. Note that with this convention the relations $P \square Q, P \doteq Q, P \ll Q$ in particular imply that there are only at most two colors among the edges running between P and Q .

Lemma 2. Let G be a vertex/edge-colored undirected graph that is identified by C^2 . If P and Q are distinct C^2 -partition classes of G , then $P \square Q, P \doteq Q, P \ll Q, Q \ll P$ or $P \equiv_3^3 Q$.

In an identified ec-POG, we call every C^2 -partition class which does not induce an undirected complete graph with only one edge color an **exception** (i.e., any class that does not fall under Item 1 of Theorem 3). Similarly, we call every pair of C^2 -partition classes P and Q for which $P \equiv_3^3 Q$ holds an **exception**.

Let G be a vertex/edge-colored graph. As before, we define the vertices of the **skeleton** S_G to be the C^2 -partition classes of G . Two distinct vertices P, Q in S_G are adjacent in S_G if the corresponding classes in G do not satisfy $P \square Q$. For identified structures, we obtain a theorem similar to Corollary 8 for graphs.

Theorem 4. Let G be a vertex-colored ec-POG. Then G is identified by C^2 if and only if the following hold: (1) Each C^2 -partition class induces a graph identified by C^2 , (2) for all C^2 -partition classes P and Q we have $P \square Q, P \doteq Q, P \ll Q, Q \ll P$ or $P \equiv_3^3 Q$, (3) the skeleton S_G is a forest, (4) there is no path P_0, P_1, \dots, P_t in S_G with $P_0 \ll P_1$ and $P_{t-1} \gg P_t$, (5) there is no path P_0, P_1, \dots, P_t in S_G where $P_0 \ll P_1$ and P_t is an exception, and (6) in every connected component of S_G there is at most one exception.

Corollary 9. Given a finite relational structure \mathfrak{A} with a universe of size n over a fixed signature we can decide in time $O(n^2 \log n)$ whether it is identified by C^2 .

Using the classification we also obtain an extension of Corollary 7 to ec-POGs and, more generally, to finite relational structures.

Corollary 10. If a finite relational structure \mathfrak{A} is identified by C^2 , then every finite relational structure obtained from \mathfrak{A} by adding unary relations is also identified by C^2 .

6 Higher Dimensions

Considering triangular graphs [7, 14] we see that for $k > 2$ any classification result for graphs identified by C^k must include an infinite number of non-trivial graphs. This already indicates that the situation for $k = 2$ is special. We show that statements analogous to Corollaries 3 and 7 do not hold for higher dimensions. For this we use the construction from [6].

Theorem 5. *For every $k > 2$, there is a graph H of size $O(k)$ identified by C^3 for which the C^k -partition is strictly coarser than the orbit partition. Moreover, not all vertex-colored versions of H are identified by C^k .*

Even if a graph is identified and the orbits are correctly determined by C^k , it may still be the case that this does not hold for all colored versions of the graph.

Theorem 6. *For every $k > 2$, there is a graph H of size $O(k)$ which is identified by C^3 such that the C^k -partition classes are the orbits of H but there are vertex-colored versions of H that are not identified by C^k and for which the C^k -partition classes are not the orbits of H .*

References

1. Arvind, V., Köbler, J., Rattan, G., Verbitsky, O.: On the power of color refinement. In: FCT 2015, (to appear 2015)
2. Atserias, A., Maneva, E.N.: Sherali-adams relaxations and indistinguishability in counting logics. *SIAM J. Comput.* **42**(1), 112–137 (2013)
3. Babai, L., Erdős, P., Selkow, S.M.: Random graph isomorphism. *SIAM J. Comput.* **9**(3), 628–635 (1980)
4. Babai, L., Kucera, L.: Canonical labelling of graphs in linear average time. In: FOCS 1979, pp. 39–46. IEEE Computer Society (1979)
5. Berkholz, C., Bonsma, P., Grohe, M.: Tight lower and upper bounds for the complexity of canonical colour refinement. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 145–156. Springer, Heidelberg (2013)
6. Cai, J., Fürer, M., Immerman, N.: An optimal lower bound on the number of variables for graph identifications. *Combinatorica* **12**(4), 389–410 (1992)
7. Chang, L.-C.: The uniqueness and nonuniqueness of the triangular association scheme. *Sci. Record.* **3**, 604–613 (1959)
8. Dawar, A., Holm, B.: Pebble games with algebraic rules. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part II. LNCS, vol. 7392, pp. 251–262. Springer, Heidelberg (2012)
9. Grädel, E., Otto, M.: On logics with two variables. *Theor. Comput. Sci.* **224**(1–2), 73–113 (1999)
10. Grädel, E., Otto, M., Rosen, E.: Two-variable logic with counting is decidable. In: LICS 1997, pp. 306–317. IEEE Computer Society (1997)
11. Grohe, M.: Finite variable logics in descriptive complexity theory. *Bull. Symbolic Log.* **4**(4), 345–398 (1998)
12. Grohe, M., Otto, M.: Pebble games and linear equations. In: Cégielski, P., Durand, A. (eds.) CSL 2012, of LIPIcs, vol. 16, pp. 289–304 (2012)
13. Hella, L.: Logical hierarchies in PTIME. *Inf. Comput.* **129**(1), 1–19 (1996)

14. Hoffman, A.J.: On the uniqueness of the triangular association scheme. *Ann. Math. Statist.* **31**(2), 492–497 (1960)
15. Kiefer, S., Schweitzer, P., Selman, E.: Graphs identified by logics with counting. CoRR, abs/1503.08792 (2015). full version of the paper
16. Kopczynski, E., Tan, T.: Regular graphs and the spectra of two-variable logic with counting. CoRR, abs/1304.0829 (2013)
17. Krebs, A., Verbitsky, O.: Universal covers, color refinement, and two-variable logic with counting quantifiers: Lower bounds for the depth. CoRR, abs/1407.3175 (2014)
18. Kucera, L.: Canonical labeling of regular graphs in linear average time. In: FOCS 1987, pp. 271–279. IEEE Computer Society (1987)
19. Lucas, E.: *Récréations mathématiques*. 2ième éd., nouveau tirage. Librairie Scientifique et Technique Albert Blanchard, Paris (1960)
20. McKay, B.D., Piperno, A.: Practical graph isomorphism, II. *J. Symb. Comput.* **60**, 94–112 (2014)
21. Otto, M.: Canonization for two variables and puzzles on the square. *Ann. Pure Appl. Logic* **85**(3), 243–282 (1997)
22. Pratt-Hartmann, I.: Complexity of the two-variable fragment with counting quantifiers. *J. Log. Lang. Inf.* **14**(3), 369–395 (2005)
23. West, D.B.: *Introduction to Graph Theory*, 2nd edn. Pearson, Upper Saddle River (2000)

Synchronizing Automata with Extremal Properties

Andrzej Kisielewicz^(✉) and Marek Szykuła

Department of Mathematics and Computer Science,
University of Wrocław, Wrocław, Poland
andrzej.kisielewicz@math.uni.wroc.pl, msz@cs.uni.wroc.pl

Abstract. We present a few classes of synchronizing automata exhibiting certain extremal properties with regard to synchronization. The first is a series of automata with subsets whose shortest extending words are of length $\Theta(n^2)$, where n is the number of states of the automaton. This disproves a conjecture that every subset in a strongly connected synchronizing automaton is cn -extendable, for some constant c , and in particular, shows that the cubic upper bound on the length of the shortest reset words cannot be improved generally by means of the extension method. A detailed analysis shows that the automata in the series have subsets that require words as long as $n^2/4 + O(n)$ in order to be extended by at least one element.

We also discuss possible relaxations of the conjecture, and propose the image-extension conjecture, which would lead to a quadratic upper bound on the length of the shortest reset words. In this regard we present another class of automata, which turn out to be counterexamples to a key claim in a recent attempt to improve the Pin-Frankl bound for reset words.

Finally, we present two new series of slowly irreducibly synchronizing automata over a ternary alphabet, whose lengths of the shortest reset words are $n^2 - 3n + 3$ and $n^2 - 3n + 2$, respectively. These are the first examples of such series of automata for alphabets of size larger than two.

1 Introduction

In this paper we deal with *deterministic finite (semi) automata (DFA)* $\mathcal{A} = (Q, \Sigma, \delta)$, where Q is a non-empty *set of states*, Σ is a non-empty *alphabet*, and $\delta: Q \times \Sigma \mapsto Q$ is the complete *transition function*. We extend δ to $Q \times \Sigma^*$ and $2^Q \times \Sigma^*$ in a natural way. The image $\delta(S, w)$ is denoted shortly by Sw , and the preimage $\delta^{-1}(S, w) = \{q \in Q \mid qw \in S\}$ is denoted by Sw^{-1} . Throughout the paper, by n we denote the cardinality $|Q|$. The *rank* of a word $w \in \Sigma^*$ is the cardinality $|Qw|$. A word w of rank 1 is called a *synchronizing word*. An automaton for which there exists a synchronizing word is called *synchronizing*.

Andrzej Kisielewicz—Supported in part by Polish MNiSZW grant IP 2012 052272.

Marek Szykuła—Supported in part by Polish NCN grant DEC-2013/09/N/ST6/01194.

The *reset length* is the length of the shortest reset words. A word w *compresses* a subset $S \subseteq Q$ if $|Sw| < |S|$, and a subset that admits such a word is called *compressible*.

The famous Černý conjecture states that every synchronizing automaton \mathcal{A} with n states has a reset word of length $\leq (n-1)^2$. This conjecture was formulated by Černý in 1964 [8], and is considered the longest-standing open problem in combinatorial theory of finite automata. So far, the conjecture has been proved only for a few special classes of automata, and the best general upper bound proven is $\frac{n^2-n}{6} - 1$ ($n \geq 4$). It is known that it is enough to prove the conjecture for strongly connected automata. We refer to [13, 21] for excellent surveys discussing recent results.

Many partial results towards the solution of the Černý conjecture use the so-called *extension method* (e.g. [4, 5, 7, 9, 12, 16–18]). The essence of this method is to look for a sequence of (short) words w_1, \dots, w_{n-1} such that $S_1 = \{q\}$ for some $q \in Q$, $S_{k+1} = S_k w_k^{-1}$, and $|S_{k+1}| > |S_k|$ for $k \geq 1$. Then, necessarily, $S_n = Q$, and the word $w_{n-1} \dots w_2 w_1$ is synchronizing. (It may happen that a shorter sequence of words does the job.) In other words, we try to extend subsets of Q , starting from a singleton, rather than to compress subsets starting from Q . In connection with this method we say that $S \subseteq Q$ is *m-extendable* if there is a word w of length at most m with $|Sw^{-1}| > |S|$. An automaton \mathcal{A} is *m-extendable*, if each subset of its states is *m-extendable*. For example, Kari [12] proved that automata whose underlying digraph is Eulerian are *n-extendable*. A simple calculation shows that such automata satisfy the Černý conjecture.

Unfortunately, it is not true, in general, that each strongly connected synchronizing automaton is *n-extendable*. In [6] Berlinkov has constructed a series of strongly connected synchronizing automata with subsets for which the shortest extending words are of length $2n-3$. Thus they are not *cn-extendable* for any constant $c < 2$. However, it remained open whether each synchronizing automaton is *2n-extendable*. The question was also posed in [13]. Its importance comes from the fact, that it would imply a quadratic upper bound for the length of the shortest reset words, and many proofs of the Černý conjecture for special cases rely on the fact that subsets are extendable by short words.

In Sect. 2 we present a series \mathcal{A}_{2m-1} of strongly connected synchronizing automata with $n = 2m - 1$ states that are not *cn-extendable* for any constant c . We show that they have subsets whose shortest extending words have length $n^2/4 + O(n)$. It follows that, in general, the extension method cannot be used to improve the cubic upper bound. On the other hand, we propose a possible weakening of the condition of *m-extendability*, which would imply quadratic upper bounds. To prove some lower bound on the constant in our conjecture, a series of automata \mathcal{B}_{2m} is presented, which turn out to be also connected with a recent attempt to improve the Pin-Frankl bound [20]. It provides counterexamples of arbitrary order to the false statement in [20, Lemma3], in addition to the single counterexample on 4 states constructed in [10].

Automata with reset lengths close to the Černý bound $(n-1)^2$ are referred to as *slowly synchronizing*. Such extremal series and particular examples of

automata are of special interest, and there are only a few of them known [1–3, 11, 15, 19]. The first known such series is the famous Černý series [8], reaching the Černý bound $(n - 1)^2$. The authors of [1] found 10 infinite series of extremal automata other than the Černý series over binary alphabets. Yet, until now, no non-trivial series of synchronizing automata with reset length $n^2 + O(n)$ have been found for larger alphabets.

Of course, we are not interested in examples that can be obtained just by adding arbitrary letter to binary slowly synchronizing automata. We call an automaton *irreducibly synchronizing*, if removing any letter yields a non-synchronizing automaton. In Sect. 4 we present two such series over a ternary alphabet, with the reset lengths $n^2 - 3n + 3$ and $n^2 - 3n + 2$, respectively.

2 A Series with Quadratically Extendable Subsets

For $m \geq 3$, let $n = 2m - 1$. Let $\mathcal{A}_{2m-1} = \langle Q_{2m-1}, \{a, b\}, \delta_{2m-1} \rangle$ be the automaton shown in Fig. 1. $Q_{2m-1} = \{q_1, \dots, q_{2m-1}\}$ and δ_{2m-1} is defined as follows:

$$\delta_{2m-1}(q_i, a) = \begin{cases} q_1, & \text{if } i = m, \\ q_{m+1}, & \text{if } i = 2m - 1, \\ q_{i+1}, & \text{otherwise,} \end{cases} \quad \delta_{2m-1}(q_i, b) = \begin{cases} q_i, & \text{if } 1 \leq i \leq m - 1, \\ q_{2m-1}, & \text{if } i = m, \\ q_m, & \text{if } i = 2m - 1, \\ q_{i-m}, & \text{otherwise.} \end{cases}$$

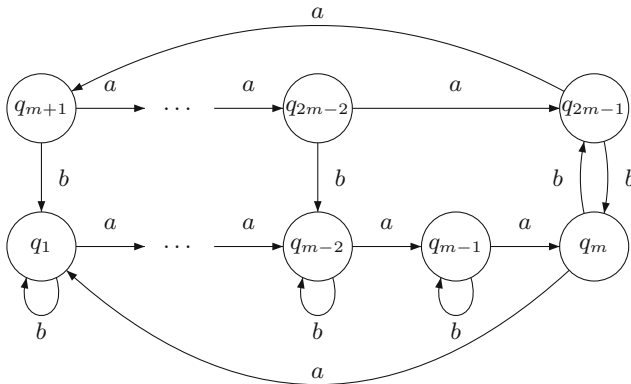


Fig. 1. The automaton \mathcal{A}_{2m-1}

It is easily verified that \mathcal{A}_{2m-1} is strongly connected. Let us define $Q_U = \{q_{m+1}, \dots, q_{2m-1}\}$ and $Q_D = \{q_1, \dots, q_m\}$. First we show that \mathcal{A}_{2m-1} is synchronizing by a word of length $2m^2 - 2m + 2 = (n^2 + 3)/2$.

Proposition 1. *The word $baba^m b(a^{m-1}ba^m b)^{m-2}$ synchronizes \mathcal{A}_{2m-1} to the state q_1 .*

Proof. First observe that $Qbaba^mb = Q_D \setminus \{q_m\}$. It suffices to ensure that if $S = \{q_1, \dots, q_i\}$ for $i = 2, \dots, m - 1$, then $Sa^{m-1}ba^mb = \{q_1, \dots, q_{i-1}\}$. Indeed, $Sa^{m-1} = \{q_1, \dots, q_{i-1}, q_m\}$; then $q_mba^mb = q_1$ and $\{q_1, \dots, q_{i-1}\}$ is mapped by the action of ba^mb to itself. \square

The following result allows us to refute the hypothesis that there exists a constant $c > 0$ such that each subset in a synchronizing automaton is cn -extendable. We first prove a weaker estimation of order $\Theta(n^2)$, because it also allows to refute the hypothesis, while it has a simpler proof.

For any $S \subseteq Q$ we say that $q_i \in S \cap Q_U$ is covered (in S) if $q_ib \in S$.

Lemma 1. *Let $S \subsetneq Q$ be a non-empty set, whose states from $S \cap Q_U$ are all covered in S . Then a shortest word w such that $|Sw^{-1} \cap Q_D| > |S|$ has length at least m .*

Proof. Let $w = a_k \dots a_1$ be a shortest word of length k such that $|Sw^{-1} \cap Q_D| > |S|$. Let $S_i = Sa_1^{-1} \dots a_i^{-1}$ for $i = 0, \dots, k$. So, $S_0 = S$, and $S_k = Sw^{-1}$.

Since the length of w is assumed to be the least possible, the action of a_k cannot be a permutation, and therefore $a_k = b$. Moreover, since the transformation induced by b maps only one state from Q_D to Q_U , namely $q_mb = q_{2m-1}$, S_{k-1} must contain q_{2m-1} , and must not contain q_m . Thus, q_{2m-1} is not covered in S_{k-1} .

For any $T \subseteq Q$, $Tb^{-2} \cap Q_D \subseteq T \cap Q_D$. It follows that, if $a_{k-1} = b$, then $|S_{k-2} \cap Q_D| = |S_{k-1} \cap Q_D|$, which contradicts the assumption about the shortest length. Hence $a_{k-1} = a$. Now, it follows that S_{k-2} contains q_{m+1} , which is uncovered. Consequently, $a_{k-2} = a$, as any preimage under b does not contain an uncovered state other than q_{2m-1} . Similarly, S_{k-3} contains uncovered state q_{m+2} (provided $m > 3$). Repeating this argument $2m - 1 - (m + 2) = m - 3$ times we have that $a_i = a$ for $k - 1 \geq i \geq k - m + 1$, and $|w| = k \geq m$ as required. \square

Theorem 1. *The shortest words extending the subset Q_U have length at least $2 + m \lceil (m - 3)/2 \rceil$.*

Proof. Let $u = a_k \dots a_1$ be a shortest word such that $|Q_U u^{-1}| > |Q_U|$. Obviously, u ends with b , that is, $a_1 = b$, since $Q_U a^{-1} = Q_U$. Also, as in the proof of Lemma 1, $a_k = b$. Moreover, $Q_U b^{-1} = \{q_m\}$.

For a subset $X \subseteq Q$ we define $d(X) = |X \cap Q_D|$. Observe that $|Xb^{-1}| \leq 2d(X) + 1$.

Let

$$T = \{q_m\}a_2^{-1} \dots a_{k-1}^{-1}.$$

Since $|Q_U u^{-1}| = |Ta_k^{-1}| \geq m$, and $a_k = b$, we have that $m \leq 2d(T) + 1$, and so, $d(T) \geq \lceil (m - 1)/2 \rceil$.

Thus, we have $u = bvb$, where $v = a_{k-1} \dots a_2$, and $\{q_m\}v^{-1} = T$. We define by induction the consecutive factors of v as follows: $v = v_\ell v_{\ell-1} \dots v_1$, $T_0 = \{q_m\}$, and for each $i = 1, 2, \dots, \ell$, $T_i = \{q_m\}v_1^{-1} \dots v_i^{-1}$, and v_i is the shortest factor of v such that $d(T_i) > d(T_{i-1})$. Since they are the shortest words whose inverse

action increases the number $d(T_{i-1})$, each v_i begins with b , and the increase is by one: $d(T_i) = d(T_{i-1}) + 1$. Moreover, for each i , all the states in $T_i \cap Q_U$ are covered in T_i .

By applying Lemma 1 for each T_i , we obtain $|v_i| \geq m$. Since $d(q_m) = 1$ and $d(T) \geq \lceil (m-1)/2 \rceil$, we have that $\ell \geq \lceil (m-3)/2 \rceil$. Thus $|u| \geq 2 + m \lceil (m-3)/2 \rceil$ as required. \square

The lower bound in the theorem above is rather rough. While it suffices to refute the conjecture in question, it is natural to ask for a better estimation. In this connection we have the following

Theorem 2. *The shortest words extending the subset Q_U have length $m^2 + O(m)$. Any non-empty proper subset $S \subset Q$ can be extended by a word of length at most $m^2 + O(m)$.*

The proof of this result is much longer and involved than the proof of Theorem 1, and therefore it is not reproduced here. We describe only its main idea.

The beginning of the proof is the same as in the proof of the previous theorem. We make use of the fact that to extend the set $\{q_m\}$ to a set of cardinality $2d$ (for some d) one needs to obtain first a set $\{q_m\}w^{-1} = X = U \cup D$ with $|D| \geq d$, $D \subseteq Q_D$, $U \subseteq Q_U$. Next, we show that to achieve this aim the most effective (using a shortest word) is the greedy procedure described below.

First we use the word $w = ba^{2m-3}ba^2$. It is easy to check that $\{q_m\}w^{-1} = \{q_1, q_m\}$. Then we use repeatedly $u = ba^{2m-1}$, obtaining the sets $\{q_1, \dots, q_i, q_m\}$ for $i = 2, 3, \dots, d$. Using the greedy strategy we can find a word extending Q_U of length $m^2 - 3m/2 + 4$, in case of even m , and $m^2 - m + 2$, in case of odd m . This yields easily the second statement. The main difficulty is to demonstrate that the greedy strategy is the most efficient one. It requires to consider other possible strategies and to introduce special terminology to handle this.

We need to remark, that the fact that the greedy strategy is the most efficient way to extend the subset $\{q_m\}$ by a chosen number of elements does not mean that it is the most efficient way to extend the subset Q_U . The problem is that after obtaining the required number of states in D , which in case of even m is exactly $m/2$, we may still need a translation of D , so that the required number of states is in $D \cap \{q_1, \dots, q_{m-2}\}$ (and applying subsequently b doubles the number of states). We have checked that sometimes different strategies to extend Q_U lead to words of the same length as the one obtained by the greedy strategy. It may happen that there are cases when a different strategy is slightly more efficient (within $O(m)$ summand).

Note also, that if the Černý conjecture is true, then every synchronizing automaton is $(n-1)^2$ -extendable, thus asymptotically no better bound is possible. However, we also believe that our series represents the worst possible case up to $O(n)$, that is, we conjecture that every strongly connected synchronizing automaton is $(n^2/4 + O(n))$ -extendable.

It is possible to define a similar series for all even $n = 2m$ (Fig. 2). The counterparts of Theorems 1 and 2 can be proved in a similar way.

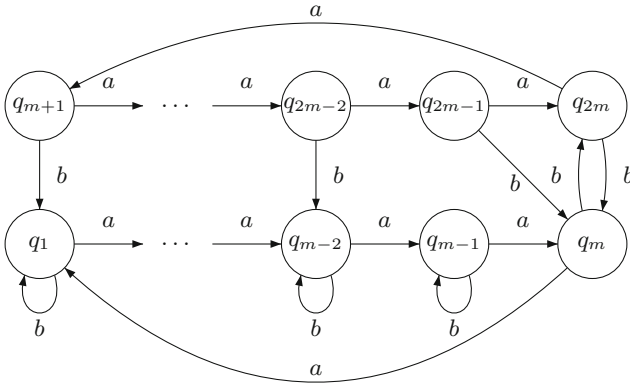


Fig. 2. The automaton \mathcal{A}_{2m}

3 Relaxing the Extension Property

In view of the examples of subsets that are not extendable by words of length n [6], some efforts were made to relax the extension conjecture (see also the discussion in [13]). For example, one could suppose that extending an easily (linearly) extendable subset cannot lead to a difficultly (quadratically) extendable subset (this is so for \mathcal{A}_{2m-1} , and in general, would allow us to prove a quadratic bound for the length of the shortest reset words). Berlinkov proposed¹ the following relaxed *conservative extension conjecture*: There exists a constant c such that if a subset $S \subset Q$ can be extended to a subset $T \subset Q$ by a word v of length at most cn ($T = Sv^{-1} \supset S$), then also T can be extended by a word u of length at most cn ($Tu^{-1} \supset T$).

This holds true for the series from [6] and also for our series \mathcal{A}_{2m-1} . However, we construct a counterexample by modifying our series; see Fig. 3. Observe that $S = \{q_{m+1}, \dots, q_{2m-1}\}$ is extended by a to $T = S \cup \{q_{2m}\}$. But then we must apply b^{-1} , resulting in $\{q_m\}$. Again, the arguments from the proof of Theorem 1 hold and we need a word of length $\Omega(n^2)$ to extend it to a subset larger than S . So in fact, this is a counterexample for the stronger statement with v of length 1 and without assuming that Tu^{-1} contains T .

3.1 Image Extending Conjecture

One may observe that the subsets Q_{2m-1} requiring long extending words are images of Q_{2m-1} under the action of no word. Hence, they cannot appear as intermediate subsets in the process of applying the consecutive letters of a synchronizing word. From the point of view of the extension method, if we consider prefixes $w_1w_2 \dots w_i$ with $|Qw_1 \dots w_i| > |Qw_1 \dots w_{i+1}|$, we only need to find

¹ First Russian-Finnish Symposium on Discrete Mathematics (RuFiDiM 2011).

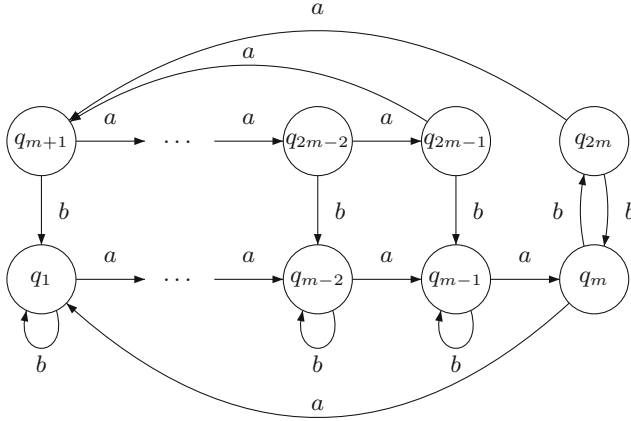


Fig. 3. An automaton with a quadratically extendable preimage

extending words for the images $Qw_1 \dots w_{i+1}$. Thus, to apply iteratively extending words starting from a singleton, at each step, the resulted preimage must be also an image, or at least it needs to contain an image of the larger cardinality than the set in question.

This restriction leads to the following weaker conjecture that we believe is true.

Conjecture 1 (Image-Extension). There exists a constant c such that, for every strongly connected synchronizing automaton $\mathcal{A} = (Q, \Sigma, \delta)$ and every non-empty $S \subset Q$ with $Qw = S$ for some $w \in \Sigma^*$, there exists a word u of length at most cn and a subset $T \subset Q$ such that $T \subseteq Su^{-1}$, $Qv = T$ for some $v \in \Sigma^*$, and $|T| > |S|$.

This conjecture implies a quadratic bound on the length of the shortest synchronizing word, yet we show that it cannot be a tool to prove the Černý conjecture because of the following

Proposition 2. *The constant c in Conjecture 1 must be at least $3/2$.*

Proof. For $n = 2m \geq 8$, consider the automaton \mathcal{B}_{2m} from Fig. 4. Clearly \mathcal{B}_{2m} is strongly connected. To show that it is synchronizing, it is sufficient to prove that every pair of states can be compressed. Let $\{p, q\}$ be a pair of distinct states. Suppose that p and q lie in different cycles of a ; without loss of generality, $p \in \{q_{m+1}, \dots, q_{2m-1}\}$ and $q \in \{q_1, \dots, q_m\}$. Since the lengths of the cycles of a are relatively prime, by a word of the form a^i we can map p, q to any other pair of states with $pa^i \in \{q_{m+1}, \dots, q_{2m-1}\}$ and $qa^i \in \{q_1, \dots, q_m\}$. In particular, we can map them to q_{2m-1} and q_m . Then b compresses this pair. If p and q lie in the upper cycle $(q_{m+1}, \dots, q_{2m-1})$ of a , then by a word of the form a^i we can map one of the states to q_{2m-2} so that the second state is not mapped to q_{2m-1} . Then using b results in a pair of states in different cycles of a , and we repeat

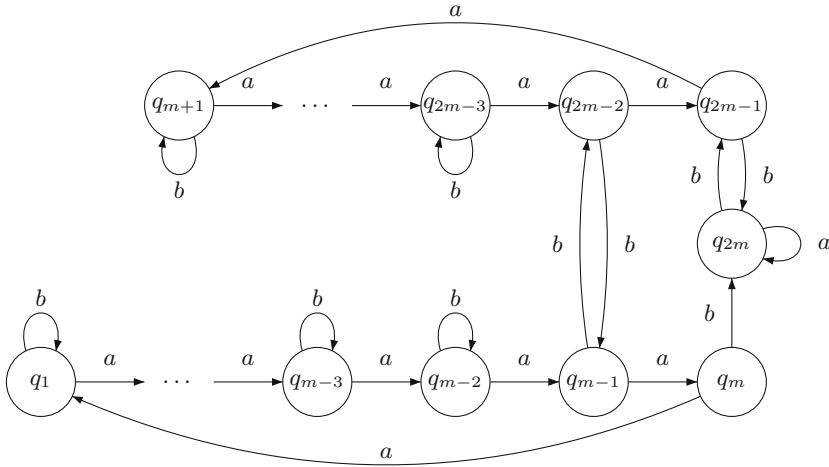


Fig. 4. The automaton \mathcal{B}_{2m}

the argument above. Similarly, if p and q lie in the lower cycle (q_1, \dots, q_m) , then we can map one of them to q_{m-1} and the second one to a state other than q_m , and again apply b . Finally, if $p = q_{2m}$ then either using b or ab (depending on q) results in a pair without q_{2m} .

Consider now $S = \{q_{m-3}, q_{m-2}\}$. Since b can reduce the size of the subset only by 1, and a is a permutation, there exists a word of rank 2. Then we can map the resulted pair $\{p, q\}$ onto S by the arguments above. Thus S is an image of Q under the action of some word.

We show that the shortest words extending S have length $\frac{3n}{2} - 1$. Let u be a shortest extending word of S . To simplify notation in the remaining part of the proof, we consider the reversed word $w = u^R$, and the inverse actions a^{-1}, b^{-1} of both letters.

Obviously w starts with a^{m-2} , since applying b earlier does not result in a new set, or results in a proper subset in the case of $\{q_m, q_1\}$. Then we have $\{q_{m-1}, q_m\}$, and we consider the following two cases:

1. The next letter is b . Here we have $\{q_{2m-2}\}$, and the next part of w is a^{m-2} , which results in $\{q_{2m-1}\}$. Then there must be b^2 , and we obtain $\{q_{2m-1}, q_m\}$. The length of w considered so far is $(m - 2) + 1 + (m - 2) + 2 = 2m - 1$.
2. The next letter is a . Here we have $\{q_{m-2}, q_{m-1}\}$, and the next letter is b . The next part of w is a^{m-2} , which results in $\{q_{2m-1}, q_m\}$. The length of w considered so far is $(m - 2) + 1 + 1 + (m - 2) = 2m - 2$.

Since w is a shortest word, the second case must take place. The next part of w must be a^{m-1} , which maps $\{q_{2m-1}, q_m\}$ to $\{q_{2m-1}, q_1\}$. Then b^2 is the shortest word extending $\{q_{2m-1}, q_1\}$, which results in $\{q_{2m-1}, q_m, q_1\}$. Therefore, the length of w is $2m - 2 + (m - 1) + 2 = 3m - 1 = 3n/2 - 1$. \square

3.2 Separating States

The series of automata \mathcal{B}_{2m} in Fig. 4 has another interesting property connected with a recent attempt to improve the Pin-Frankl bound [20]. Lemma 3 in the above mentioned paper, claiming that for every state q there exists a word w of length at most n such that $q \notin Qw$, turned out to be false. In a recent short note [10] the authors present a certain automaton on 4 states and show that it is a counterexample to the statement in [20, Lemma3]. In this connection it is worth observing that our series \mathcal{B}_{2m} provides counterexamples of arbitrary order.

It is not difficult to prove the following

Proposition 3. *The shortest words w such that $q_{2m} \notin Qw$ are of length $2m + 2 = n + 2$.*

The length w in our proposition exceeds n only by 2. It is interesting whether there are counterexamples requiring still longer words than those of length $n + 2$. The open question in [10] asks whether there exists a constant c such that, for any finite automaton and its state q , there exists a word w of length not greater than cn such that $q \notin Qw$.

4 Slowly Synchronizing Automata on a Ternary Alphabet

In this section we present two series of ternary irreducibly slowly synchronizing automata. They turn out to be related to the digraph W_n defined in [2]. They are of interest since so far only such series over a binary alphabet have been known.

Let $Q_n = \{q_1, \dots, q_n\}$, $n \geq 3$, and $\Sigma = \{a, b, c\}$. Let $\mathcal{M}_n = \langle Q_n, \Sigma, \delta_n \rangle$ and $\mathcal{M}'_n = \langle Q_n, \Sigma, \delta'_n \rangle$ be the automata shown in Figure 5. The transition functions δ_n and δ'_n are defined as follows:

$$\delta(q_i, a) = \begin{cases} q_{i+1}, & \text{if } 1 \leq i \leq n - 1, \\ q_2, & \text{if } i = n, \end{cases} \quad \delta(q_i, c) = \begin{cases} q_n, & \text{if } i = 1, \\ q_i, & \text{if } 2 \leq i \leq n - 1, \\ q_1, & \text{if } i = n, \end{cases}$$

$$\delta(q_i, b) = \begin{cases} q_2, & \text{if } i = 1, \\ q_i, & \text{if } 2 \leq i \leq n - 1, \end{cases} \quad \delta'(q_i, c) = \begin{cases} q_i, & \text{if } 1 \leq i \leq n - 1, \\ q_1, & \text{if } i = n, \end{cases}$$

and $\delta'(q_i, a) = \delta(q_i, a)$, $\delta'(q_i, b) = \delta(q_i, b)$, otherwise.

In determining the reset length of the series we applied a technique, which is alternative to that of [1, 2]. Our method is based on analyzing the behavior of the inverse BFS algorithm finding the length of the shortest reset words [14] and is suitable, in general, for a more mechanical way to establish the reset lengths of concrete automata. Also, it may lead to simpler proofs in case of larger alphabets (when the number of induced automata by combinations of letters is large). It relies on the fact, that when we are applying the inverse actions of letters starting from all singletons, then the number of new resulted sets is always bounded by a

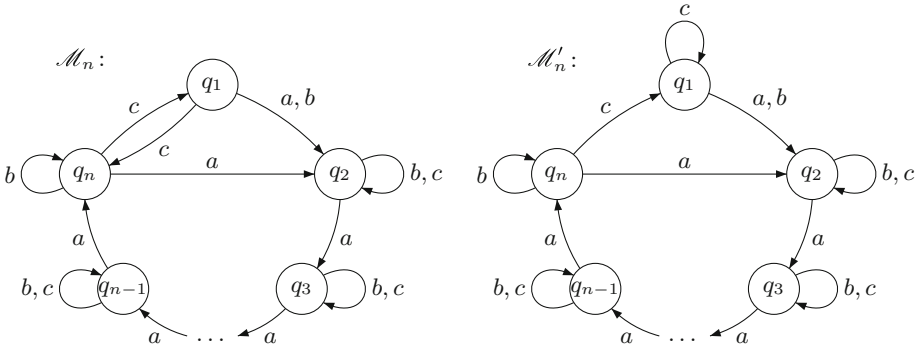


Fig. 5. \mathcal{M}_n with reset length $n^2 - 3n + 3$, and \mathcal{M}'_n with reset length $n^2 - 3n + 2$

very small constant. In fact, a particular form of this method was first used for the Černý series [8]. Interestingly, this method also works fine in a very similar way for all slowly synchronizing series defined in [1-3], as they all have this property.

For a synchronizing automaton $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ with $n > 1$ states, we define the sequence of families (L_i) of the subsets of Q . Let L_0 be the family of all the singletons, which are a common end of more than one edge with the same label. We will define L_i inductively for $i \geq 1$. Let $L'_i = \{Sa^{-1} : S \in L_{i-1}, a \in \Sigma\}$. A set $S \in L'_i$ is called *visited*, if $|S| = 1$ or there is $T \in L_j, T \supseteq S$ for $j < i$, or there is $T \in L'_i, T \supsetneq S$. We define L_i to be the set of all non-visited sets from L'_i .

The proof of the following lemma in a more general form can be found in [14, Theorem1].

Lemma 2. *There exists a shortest reset word w such that for any suffix u of w of length i , $\{q\}u^{-1} \in L_i$ for some $q \in Q$. The smallest i such that $Q \in L_i$ is the reset length of the automaton.*

Theorem 3. *For $n \geq 3$, the automata \mathcal{M}_n and \mathcal{M}'_n are irreducibly synchronizing. The first has reset length $n^2 - 3n + 3$, and the second has reset length $n^2 - 3n + 2$.*

Proof. One easily verifies that the word $acb(a^{n-2}cb)^{n-3}$ synchronizes \mathcal{M}_n , and has the length $3 + (n - 2 + 2)(n - 3) = n^2 - 3n + 3$. Also the word $cb(a^{n-2}cb)^{n-3}$ synchronizes \mathcal{M}'_n , and has the length $n^2 - 3n + 2$.

We show that there is no shorter reset word for \mathcal{M}_n . By Lemma 2, it is sufficient to show what is the smallest i such that $Q_n \in L_i$, which is the length of the shortest reset words. Here only q_2 is a common end of more than one edge with the same label, so $L_0 = \{\{q_2\}\}$.

We claim that for each i with $0 \leq i \leq n - 3$, $L_{in} = \{\{q_2, \dots, q_{2+i}\}\}$, $L_{(i-1)n+4} = \{\{q_{n-2}, q_{n-1}, q_n, q_1, \dots, q_{i-1}\}\}$ if $2 \leq i \leq n - 3$, and $L_{(i-1)n+4} = L_4 = \{\{q_{n-2}, q_{n-1}\}\}$ if $i = 1$. The proof follows by induction. Clearly for $i = 0$

the claim holds. Consider some i , and assume that the claim holds for $i' \leq i$, so $L_{in} = \{\{q_2, \dots, q_{2+i}\}\}$. We will show the claim for $i + 1$, that is, for $L_{(i+1)n}$ and L_{in+4} .

The action of c^{-1} for the set from L_{in} results in the same set, so we have $L_{in+1} = \{S_1, T_1\}$, where $S_1 = \{q_n, q_1, \dots, q_{1+i}\}$ was obtained by the action of a^{-1} and $T_1 = \{q_1, \dots, q_{2+i}\}$ by the action of b^{-1} . Consider the sets obtained from S_1 . Observe that if a set contains both q_n and q_1 , then only the action of a^{-1} can result in a non-visited set. If $i = 0$ then $S_1 a^{-1} = \{q_n, q_1\} a^{-1} = \{q_{n-1}\}$ is a visited singleton. If $i \geq 1$ then $S_2 = S_1 a^{-1} = \{q_{n-1}, q_n, q_1, \dots, q_i\}$. But $S_2 a^{-1}$ is $\{q_{n-2}, q_{n-1}\}$ if $i = 1$, or $\{q_{n-2}, q_{n-1}, q_n, q_1, \dots, q_{i-1}\}$ if $i \geq 2$; so $S_2 a^{-1}$ is visited by the assumption of $L_{(i-1)n+4}$. Consider the sets obtained from T_1 . Only $T_2 = T_1 c^{-1} = \{q_n, q_2, \dots, q_{2+i}\}$ is non-visited. Then let $T_3 = T_2 a^{-1} = \{q_{n-1}, q_n, q_1, \dots, q_{1+i}\}$. For $T_2 b^{-1} = \{q_n, q_1, \dots, q_{2+i}\}$ observe that in the next step it results either in T_3 or in itself. Only the action of a^{-1} applied to T_3 results in a non-visited set, so $L_{in+4} = \{\{q_{n-2}, q_{n-1}\}\}$ if $i = 0$, and $L_{in+4} = \{\{q_{n-2}, q_{n-1}, q_n, q_1, \dots, q_i\}\}$ if $i \geq 1$. Now, if $i = 0$ then only the action of a^{-1} results in a non-visited set over the next $n - 4$ steps resulting in $L_n = \{q_2, q_3\}$. Similarly, if $i \geq 1$ then by the next $i - 1$ steps by the action of a^{-1} we have $\{q_{n-1-i}, \dots, q_n, q_1\}$. Again, through the next $n - 3 - i$ steps only the action of a^{-1} results in a non-visited set, and we finally have $L_{(i+1)n} = \{\{q_2, \dots, q_{3+i}\}\}$.

From the claim it follows that Q_n does not appear in L for $i \leq (n - 3)n$, and $L_{(n-3)n} = \{\{q_2, \dots, q_{n-1}\}\}$. Then applying $(acb)^{-1}$ or $(bcb)^{-1}$ results in Q_n , and there is no shorter such word as is easily verified. Hence $L_{(n-3)n+3} = \{\{Q_n\}\}$ and so $i = (n - 3)n + 3 = n^2 - 3n + 3$ is the length such that $Q_n \in L_i$.

The proof for the automaton \mathcal{M}'_n follows exactly in the same way, with the following two exceptions: $T_2 = \{q_n, q_1, \dots, q_{2+i}\}$, and finally we apply $(cb)^{-1}$ to the set from $L_{(n-3)n}$, resulting in Q_n .

It remains to show that removing any letter in \mathcal{M}_n (\mathcal{M}'_n) results in a non-synchronizing automaton. Indeed, removing the letter a results in unconnected states q_3, \dots, q_{n-1} . The only compressible pairs of states are $\{q_1, q_n\}$ under a , and $\{q_1, q_2\}$ under b . Observe that it is not possible to map the pair $\{q_{n-1}, q_n\}$ to $\{q_1, q_n\}$: Only a maps a state from $Q_n \setminus \{q_1, q_n\}$ to $\{q_1, q_n\}$, and it maps exactly one such state. However both q_1 and q_n are mapped by a to $Q_n \setminus \{q_1, q_n\}$. Hence removing the letter b results in a non-synchronizing automaton. Removing the letter c makes q_1 unreachable from the other states, hence no pair can be compressed except $\{q_1, q_n\}$ and $\{q_1, q_2\}$. \square

It seems that when we admit more letters in the alphabet, then it is more difficult to find any series of irreducible strongly connected slowly synchronizing automata. Although there are many binary series with the reset length $n^2 + O(n)$ (obtained by modifying the series from [1, 2]), it is only \mathcal{M}_n and \mathcal{M}'_n that are known over a ternary alphabet, and no such series is known over a larger alphabet. Thus it becomes an interesting problem to find more such series, if they exist at all.

Acknowledgment. We are grateful to Jakub Kowalski and anonymous referees for careful proofreading.

References

1. Ananichev, D., Gusev, V., Volkov, M.: Slowly synchronizing automata and digraphs. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 55–65. Springer, Heidelberg (2010)
2. Ananichev, D.S., Volkov, M.V., Gusev, V.V.: Primitive digraphs with large exponents and slowly synchronizing automata. *J. Math. Sci.* **192**(3), 263–278 (2013)
3. Ananichev, D.S., Volkov, M.V., Zaks, Y.I.: Synchronizing automata with a letter of deficiency 2. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 433–442. Springer, Heidelberg (2006)
4. Béal, M.P., Berlinkov, M.V., Perrin, D.: A quadratic upper bound on the size of a synchronizing word in one-cluster automata. *Int. J. Foundations Comput. Sci.* **22**(2), 277–288 (2011)
5. Berlinkov, M., Szykuła, M.: Algebraic synchronization criterion and computing reset words. In: Italiano, G.F., et al (eds.) MFCS 2015. Lecture Notes in Computer Science, vol. 9234, pp. 103–115 (2015)
6. Berlinkov, M.V.: On a conjecture by Carpi and D’Alessandro. *Int. J. Foundations Comput. Sci.* **22**(7), 1565–1576 (2011)
7. Berlinkov, M.V.: Synchronizing quasi-eulerian and quasi-one-cluster automata. *Int. J. Foundations Comput. Sci.* **24**(6), 729–745 (2013)
8. Černý, J.: Poznámka k homogénnym experimentom s konečnými automatami. *Matematicko-fyzikálny Čas. Slovenskej Akad. Vied* **14**(3), 208–216 (1964). in Slovak
9. Dubuc, L.: Sur les automates circulaires et la conjecture de Černý. *Informatique théorique et Appl.* **32**, 21–34 (1998). in French
10. Gonze, F., Jungers, R.M., Trahtman, A.N.: A note on a recent attempt to improve the Pin-Frankl bound. *Discrete Math. Theoret. Comput. Sci.* **17**(1), 307–308 (2015)
11. Gusev, V.V., Pribavkina, E.V.: Reset thresholds of automata with two cycle lengths. In: Holzer, M., Kutrib, M. (eds.) CIAA 2014. LNCS, vol. 8587, pp. 200–210. Springer, Heidelberg (2014)
12. Kari, J.: Synchronizing finite automata on Eulerian digraphs. *Theoret. Comput. Sci.* **295**(1–3), 223–232 (2003)
13. Kari, J., Volkov, M.V.: Černý’s conjecture and the road coloring problem. In: *Handbook of Automata*. European Science Foundation (2013, to appear)
14. Kisielewicz, A., Kowalski, J., Szykuła, M.: Computing the shortest reset words of synchronizing automata. *J. Comb. Optim.* **29**(1), 88–124 (2015)
15. Roman, A.: A note on Černý conjecture for automata over 3-letter alphabet. *J. Automata Lang. Comb.* **13**(2), 141–143 (2008)
16. Rystsov, I.K.: Quasioptimal bound for the length of reset words for regular automata. *Acta Cybernetica* **12**(2), 145–152 (1995)
17. Steinberg, B.: The averaging trick and the Černý conjecture. *Int. J. Foundations Comput. Sci.* **22**(7), 1697–1706 (2011)
18. Steinberg, B.: The Černý conjecture for one-cluster automata with prime length cycle. *Theoret. Comput. Sci.* **412**(39), 5487–5491 (2011)
19. Trahtman, A.N.: An efficient algorithm finds noticeable trends and examples concerning the Černý conjecture. In: Kráľovič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 789–800. Springer, Heidelberg (2006)

20. Trahtman, A.N.: Modifying the upper bound on the length of minimal synchronizing word. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 173–180. Springer, Heidelberg (2011)
21. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)

Ratio and Weight Quantiles

Daniel Krähmann^(✉), Jana Schubert, Christel Baier, and Clemens Dubslaff

Faculty of Computer Science, Technische Universität Dresden, Dresden, Germany
{kraehmann,schubert,baier,dubslaff}@tcs.inf.tu-dresden.de

Abstract. Several types of weighted-automata models and formalisms to specify and verify constraints on accumulated weights have been studied in the past. The lack of monotonicity for weight functions with positive and negative values as well as for ratios of the accumulated values of non-negative weight functions renders many verification problems to be undecidable or computationally hard. Our contribution comprises polynomial-time algorithms for computing ratio and weight quantiles in Markov chains, which provide optimal bounds guaranteed almost surely or with positive probability on, e.g., cost-utility ratios or the energy conversion efficiency.

1 Introduction

Markov decision processes (MDPs) and Markov chains with weight assignments are widely used for modeling resource-aware systems. A quantitative analysis in terms of probabilistic model checking (PMC) allows to reason about performance, dependability and reliability properties, e.g., the expected energy costs or the utility gained until a goal is reached. Classical logics, such as extensions of probabilistic computation-tree logic (PCTL) [1, 17], focus on non-negative weight assignments and enjoy broad tool support, e.g., by probabilistic model checkers PRISM [24] and MRMC [27]. To model a device where a battery is recharged and drained or to reason about cost-utility ratios, these classical approaches are not sufficient. However, accumulated (not necessarily positive) weights along paths or ratios thereof are no longer monotonic, which makes the design of algorithms much harder. It is hence not surprising that several undecidability results [6, 7] show limitations on the algorithmic analysis of weighted models. This motivated the research on specialized models such as *energy games* [14, 15, 25]. In the probabilistic setting of energy games, one natural task is to compute the minimal initial energy budget required to almost surely ensure an ω -regular property while not running out of energy, i.e., the accumulated weight is always positive. This computation can be done in pseudo-polynomial time [14].

The authors are supported by the DFG through the collaborative research centre HAEC (SFB 912), the Excellence Initiative by the German Federal and State Governments (cluster of excellence cfaED and Institutional Strategy), the Graduiertenkolleg QuantLA (1763), Deutsche Telekom Stiftung, the EU-FP-7 grant MEALS (295261).

In this paper, we establish polynomial-time computation schemes for weight and ratio quantiles in finite-state Markov chains. Let **wgt** stand for the accumulated value of a weight function and let **ratio** stand for the quotient of accumulated values of two non-negative weight functions. Then, our main contribution is stated by the following theorem, where ∇ is either the temporal modality \square (invariance), \diamond (reachability), $\diamond\square$ (persistence), or $\square\diamond$ (repeated reachability). More details on the notations are explained later in Sect. 2.

Theorem 1. *When φ is a Rabin or Streett side constraint, the following quantiles with $\nabla \in \{\diamond, \square, \diamond\square, \square\diamond\}$ are computable in polynomial time:*

- (i) $\text{Qu}_{\varphi}^{>0}[\nabla\text{wgt}] = \sup\{z \in \mathbb{Z} : \Pr(\nabla(\text{wgt} > z) \wedge \varphi) > 0\}$,
- (ii) $\text{Qu}_{\varphi}^{=1}[\nabla\text{wgt}] = \sup\{z \in \mathbb{Z} : \Pr(\nabla(\text{wgt} > z) \wedge \varphi) = 1\}$,
- (iii) $\text{Qu}_{\varphi}^{>0}[\nabla\text{ratio}] = \sup\{q \in \mathbb{Q} : \Pr(\nabla(\text{ratio} > q) \wedge \varphi) > 0\}$,
- (iv) $\text{Qu}_{\varphi}^{=1}[\nabla\text{ratio}] = \sup\{q \in \mathbb{Q} : \Pr(\nabla(\text{ratio} > q) \wedge \varphi) = 1\}$.

Intuitively, weight quantiles stand for optimal thresholds which can be guaranteed with positive probability (i) or almost surely (ii), e.g., for optimal battery levels achieved at some point ($\nabla = \diamond$), at any point ($\nabla = \square$), at any point after an initialization phase ($\nabla = \diamond\square$), or at infinitely many moments ($\nabla = \square\diamond$) during an execution of the system. Similar thresholds are established by positive (iii) and almost-sure (iv) ratio quantiles, e.g., guarantees on the cost-utility ratio such as the energy conversion efficiency indicated through the ratio between energy put into the system and converted into something useful. The latter quantitative measures could be used to compare engines or computer hardware concerning their energy-transmission properties. Also deviations of such quantiles, e.g., the value $|\text{Qu}_{\varphi}^{>0}[\nabla\text{ratio}] - \text{Qu}_{\varphi}^{=1}[\nabla\text{ratio}]|$, could serve as a measure of dispersion, where high dispersion indicates great adaptivity, and low dispersion may come along with stability and robustness of the system.

As we show in this paper, the computation of the quantiles in Theorem 1 relies on algorithms to decide whether $\nabla(\text{wgt} > z) \wedge \varphi$ (or $\nabla(\text{ratio} > q) \wedge \varphi$, respectively) holds almost surely or with positive probability. Using results from basic random-walk theory and variants of shortest-path algorithms, we establish polynomial-time decision procedures for these problems. Whereas solving the weight decision problems within a simple binary search turns out to be sufficient to compute weight quantiles in polynomial time, this is not the case for ratio quantiles. Here, our solution relies on establishing a finite, but exponentially large set of rational-valued candidates for the ratio quantiles and on solving a *best-approximation problem* using the so-called *continued-fraction method* [22].

Related Work. Algorithms to compute quantiles in Markovian models with non-negative weight functions have been investigated for weight-bounded reachability properties in [2, 23, 32]. In the qualitative case, [32] established a polynomial-time algorithm based on shortest-path algorithms. For Markov chains with weight functions as we consider them in this paper containing both, negative and positive values, a naïve reduction to unit-weight Markov chains exists but causes an exponential blow-up [3, 10]. In Table 1, we summarized upper complexity bounds for

Table 1. Known and new complexity bounds for weight decision problems.

Problem	Unit weight	Unit multi-weight	Weight	Multi-weight
$\Pr(\Box(\text{wgt}>z)) > 0$	PTime [10,12]	ExpTime [12]	ExpTime [10,12] PTime	2ExpTime [12] ExpSpace-complete
$\Pr(\Box(\text{wgt}>z)) = 1$	PTime [3]			

weight decision problems already established for unit weight models or through the reduction mentioned above. Complexity bounds shown in this paper are in bold. Concerning the interplay between multiple weight functions, several authors proposed linear-programming techniques for the synthesis of strategies satisfying multiple mean-payoff objectives [9], Boolean combinations of PCTL-like probability and expectation constraints [18], or constraints on the ratio of accumulated weights [33] and its long-run behavior. Known mean-payoff objectives [9,30] over one weight function `wgt` are tightly connected to $\Diamond\Box$ -ratio objectives we consider in this paper. This can be seen by choosing the weight function `wgt` as nominator and a simple step counter as the denominator of a $\Diamond\Box$ -ratio objective (see also Remark 3).

Organization. Section 2 summarizes basic concepts on weighted Markov chains. In Sect. 3 we develop polynomial-time algorithms for solving ratio and weight decision problems used to prove our main contribution (Theorem 1). The proof itself is issued in Sect. 4. We close with concluding remarks in Sect. 5. Omitted proofs can be found in the technical report [28].

2 Theoretical Foundations

The reader is supposed to be familiar with standard concepts of temporal logic, model checking and basic results from finite Markov-chain theory (see, e.g., [5,16,21,26]). We briefly summarize our notations and recall some basic concepts used throughout the paper.

Markov Chains and Path Notations. A *Markov chain* is a tuple $\mathcal{M} = (S, \iota, P)$, where S is non-empty finite set of states, $\iota \in S$ is an *initial state*, and $P: S \times S \rightarrow [0, 1]$ is a *transition probability matrix* for which $\sum_{s' \in S} P(s, s') = 1$ for all $s \in S$. Note that within this definition, we suppose that all states $s \in S$ are reachable from ι . A *transition* is a pair $(s, s') \in S \times S$ where $P(s, s') > 0$. Non-empty finite (infinite) sequences of states comprised of consecutive transitions are called finite (infinite) *paths*. The set of all finite (infinite) paths of \mathcal{M} is denoted by $\text{FinPaths}^{\mathcal{M}}$ ($\text{InfPaths}^{\mathcal{M}}$, respectively). $\text{FinPaths}_s^{\mathcal{M}}$ and $\text{InfPaths}_s^{\mathcal{M}}$ stand for the corresponding sets of paths starting in state s . The *length* of a finite path $\hat{\pi}$, denoted by $|\hat{\pi}|$, is defined as the number of transitions taken in $\hat{\pi}$, e.g., $|s_0 s_1 s_2| = 2$. The first and the last state of a finite path $\hat{\pi}$ are denoted by $\text{first}(\hat{\pi})$ and $\text{last}(\hat{\pi})$, respectively. A *cycle* ϑ is a finite path where $|\vartheta| > 0$ and $\text{first}(\vartheta) = \text{last}(\vartheta)$. We call a finite path $\hat{\pi}$ *simple*, if $\hat{\pi}$ visits each state at most once. A cycle $\vartheta = \hat{\pi} \text{first}(\hat{\pi})$

is *simple* if $\hat{\pi}$ is. We often use standard notions of temporal logics to describe events (i.e., measurable sets of paths) or state properties in Markov chains. For example, given a set C of states in \mathcal{M} , $\diamond C$ denotes the event to reach some state in C eventually, i.e., the set of all infinite paths containing a state of C . We write $s \models \exists \diamond C$ to denote that some state in C is reachable from s . For a Markov chain $\mathcal{M} = (S, \iota, P)$ we consider ω -regular properties over a set of atomic propositions AP , linked to \mathcal{M} through a labeling function $L: S \rightarrow 2^{\text{AP}}$.

Probability Space and Steady-State Probability. The probability space associated to \mathcal{M} and a state s formalizes the intuitive notion of probabilities of measurable sets of sample runs in \mathcal{M} . Given a finite path $\hat{\pi} = s_0 s_1 \dots s_n$, the *cylinder set of $\hat{\pi}$* , denoted by $\text{Cyl}(\hat{\pi})$, consists of all infinite paths of \mathcal{M} having $\hat{\pi}$ as a prefix. Cylinder sets constitute a basis of a σ -algebra on $\text{InfPaths}_s^{\mathcal{M}}$, on which $\text{Pr}_s^{\mathcal{M}}$ is defined as the unique probability measure where $\text{Pr}_s^{\mathcal{M}}(\text{Cyl}(\hat{\pi})) = P(s_0, s_1) \cdot P(s_1, s_2) \cdot \dots \cdot P(s_{n-1}, s_n)$ if $s_0 = s$ and $\text{Pr}_s^{\mathcal{M}}(\text{Cyl}(\hat{\pi})) = 0$ otherwise. For $\text{Pr}_\iota^{\mathcal{M}}$ we often write simply Pr . A *bottom strongly connected component (BSCC)* of \mathcal{M} is a set of states \mathcal{C} , where all states of \mathcal{C} can reach each other but none of the states in $S \setminus \mathcal{C}$. It is well-known that almost all paths of a Markov chain eventually enter a BSCC \mathcal{C} and visit all states of \mathcal{C} infinitely often. For almost all infinite paths π that enter \mathcal{C} , the frequency of visiting state s is the *steady-state probability* $\mathbb{S}_{\mathcal{C}}(s)$ of s in \mathcal{C} , defined as $\mathbb{S}_{\mathcal{C}}(s) = \lim_{n \rightarrow \infty} |\{k \in \{0, \dots, n\} : \pi[k] = s\}| / (n + 1)$. The vector $(\mathbb{S}_{\mathcal{C}}(s))_{s \in \mathcal{C}}$ is the unique solution of the flow equations $\sum_{s' \in \mathcal{C}} P(s, s') \cdot \mathbb{S}_{\mathcal{C}}(s') = \mathbb{S}_{\mathcal{C}}(s)$ with the side constraint $\sum_{s' \in \mathcal{C}} \mathbb{S}_{\mathcal{C}}(s') = 1$, which are computable in time polynomial in the size of \mathcal{M} .

Weighted and Energy-Utility Markov Chains. We refer to $(\mathcal{M}, \text{wgt})$ as a *weighted Markov chain* if \mathcal{M} is a Markov chain and $\text{wgt}: S \times S \rightarrow \mathbb{Z}$ is a *weight function*. The *accumulated weight* of a finite path $\hat{\pi} = s_0 s_1 s_2 \dots s_n$ is defined by $\text{wgt}(\hat{\pi}) = \text{wgt}(s_0, s_1) + \dots + \text{wgt}(s_{n-1}, s_n)$. Hence, $\text{wgt}(s) = 0$ for every $s \in S$. A cycle ϑ is said to be *positive* if $\text{wgt}(\vartheta) > 0$. Likewise, a cycle ϑ is *negative* if $\text{wgt}(\vartheta) < 0$. Given $z \in \mathbb{Z}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$, the event $\square(\text{wgt} \bowtie z)$ is defined as the set of paths $\pi = s_0 s_1 \dots \in \text{InfPaths}^{\mathcal{M}}$ where

$$\pi \models \square(\text{wgt} \bowtie z) \quad \text{iff} \quad \text{wgt}(s_0 s_1 \dots s_n) \bowtie z \text{ for all } n \in \mathbb{N}_{>0}.$$

Analogously, $\pi \models \diamond(\text{wgt} \bowtie z)$ iff $\text{wgt}(s_0 s_1 \dots s_n) \bowtie z$ for some $n \in \mathbb{N}_{>0}$, and $\pi \models \diamond \square(\text{wgt} \bowtie z)$ iff there exists $k \in \mathbb{N}$ such that $\text{wgt}(s_0 s_1 \dots s_n) \bowtie z$ for all $n \in \mathbb{N}_{\geq k}$, and $\pi \models \square \diamond(\text{wgt} \bowtie z)$ iff there exist infinitely many $n \in \mathbb{N}$ with $\text{wgt}(s_0 s_1 \dots s_n) \bowtie z$. In the remainder of the paper the modality ∇ stands for one of the temporal operators $\diamond, \square, \diamond \square, \square \diamond$.

An *energy-utility Markov chain* is a tuple $(\mathcal{M}, \text{energy}, \text{utility})$ consisting of a Markov chain \mathcal{M} with two weight functions $\text{energy}: S \times S \rightarrow \mathbb{N}_{>0}$ and $\text{utility}: S \times S \rightarrow \mathbb{N}$. The *energy-utility ratio*, briefly called *ratio*, is defined as the function $\text{ratio}: \text{FinPaths}^{\mathcal{M}} \rightarrow \mathbb{Q}_{\geq 0}$ given by $\text{ratio}(\hat{\pi}) = \text{utility}(\hat{\pi}) / \text{energy}(\hat{\pi})$, where $|\hat{\pi}| > 0$, and $\text{ratio}(\hat{\pi}) = 0$, otherwise. As before, we use LTL-like notations to describe events $\nabla(\text{ratio} \bowtie q)$ for a given rational number q . For instance, $\pi \models \square(\text{ratio} \bowtie q)$ iff $\text{ratio}(s_0 s_1 \dots s_n) \bowtie q$ for every $n \in \mathbb{N}_{>0}$ where $\pi = s_0 s_1 \dots \in \text{InfPaths}^{\mathcal{M}}$.

Expected Weight and Long-Run Ratio. Given a BSCC \mathcal{C} of an weighted Markov chain $(\mathcal{M}, \text{wgt})$, the *expected weight of \mathcal{C}* is defined as $\mathbb{E}_{\mathcal{C}}(\text{wgt}) = \sum_{s, s' \in \mathcal{C}} \text{wgt}(s, s') \cdot \mathbb{S}_{\mathcal{C}}(s) \cdot P(s, s')$. It is well-known that for almost all infinite paths $\pi = s_0 s_1 \dots$ of an energy-utility Markov chain $(\mathcal{M}, \text{energy}, \text{utility})$ the *long-run ratio* $\mathbb{L}(\text{ratio})(\pi) = \lim_{n \rightarrow \infty} \text{ratio}(s_0 s_1 \dots s_n)$ is well-defined. That is, almost all paths π eventually reaching a BSCC \mathcal{C} have the same long-run ratio, namely $\mathbb{L}_{\mathcal{C}}(\text{ratio}) = \mathbb{E}_{\mathcal{C}}(\text{utility})/\mathbb{E}_{\mathcal{C}}(\text{energy})$. We refer to $\mathbb{L}_{\mathcal{C}}(\text{ratio})$ as the *long-run ratio of \mathcal{C}* . Observe that $\mathbb{L}_{\mathcal{C}}(\text{ratio})$ is rational and can be computed in time polynomial in the size of \mathcal{M} and the encoding length of energy and utility [4].

3 Ratio and Weight Decision Problems

To establish Theorem 1, we rely on the following theorem to whose proof we dedicate this section.

Theorem 2. *When φ is a Rabin or Streett side constraint, the following decision problems can be solved in polynomial time:*

- (i) $\Pr(\nabla(\text{wgt} > z) \wedge \varphi) > 0$,
- (ii) $\Pr(\nabla(\text{wgt} > z) \wedge \varphi) = 1$,
- (iii) $\Pr(\nabla(\text{ratio} > q) \wedge \varphi) > 0$,
- (iv) $\Pr(\nabla(\text{ratio} > q) \wedge \varphi) = 1$.

First, we give a graph-based characterization for the stated problems without the side constraint φ . Then, we generalize our results for arbitrary ω -regular side constraints and sketch a polynomial-time decision procedure.

Ratio-Weight Transformation. Ratio constraints for energy-utility Markov chains are reducible to weight constraints in weighted Markov chains [3, 7]: Given an energy-utility Markov chain $(\mathcal{M}, \text{energy}, \text{utility})$ and $q \in \mathbb{Q}_{\geq 0}$, there exists a weight function wgt_q for \mathcal{M} such that the events $\nabla(\text{ratio} \bowtie q)$ and $\nabla(\text{wgt}_q \bowtie 0)$ with $\bowtie \in \{<, \leq, =, \geq, >\}$ coincide. The definition of wgt_q is as follows: Suppose $q = a/b$ with $a, b \in \mathbb{N}$, then $\text{wgt}_q: S \times S \rightarrow \mathbb{Z}$ is given by $\text{wgt}_q(s_1, s_2) = b \cdot \text{utility}(s_1, s_2) - a \cdot \text{energy}(s_1, s_2)$. Hence, it suffices to consider weight constraints.

Graph-Based Characterization. We focus on the event $\square(\text{wgt} > z)$ and provide a graph-based characterization for $\Pr(\square(\text{wgt} > z)) > 0$, which crucially relies on the expected weight of BSCCs. Similar characterizations can be obtained for $\Pr(\square \diamond (\text{wgt} > z)) > 0$ and $\Pr(\square \diamond (\text{wgt} > z)) = 1$. The case $\Pr(\square(\text{wgt} > z)) = 1$ is straight-forward [3] and the other cases for the operators \diamond and $\diamond \square$ can be shown through basic dualities.

In the following, let $\mathcal{M} = (S, \iota, P, \text{wgt})$ be a weighted Markov chain and $z \in \mathbb{Z}$ be a threshold. When a BSCC \mathcal{C} contains negative cycles and $\mathbb{E}_{\mathcal{C}}(\text{wgt}) < 0$ or $\mathbb{E}_{\mathcal{C}}(\text{wgt}) = 0$, any path reaching \mathcal{C} does not contribute to the probability mass $\Pr(\square(\text{wgt} > z))$, i.e., $\Pr(\square(\text{wgt} > z) \wedge \diamond \mathcal{C}) = 0$. For the remaining BSCCs \mathcal{C} we can state conditions under which $\Pr(\square(\text{wgt} > z) \wedge \diamond \mathcal{C})$ is positive. To establish such further conditions, we employ the following notations.

The *constraint-distance function* $cdist: S \times \mathbb{Z} \rightarrow \mathbb{Z} \cup \{\pm\infty\}$ can be seen as constraint variant of standard longest-paths: $cdist(s, z) = \sup_{\hat{\pi}} \text{wgt}(\hat{\pi})$ where the supremum ranges over all finite paths $\hat{\pi}$ starting from ι and ending in s such that $\text{wgt}(\rho) > z$ for all prefixes ρ of $\hat{\pi}$ with $|\rho| \geq 1$. The *minimal-credit function* $\mu: S \rightarrow \mathbb{N} \cup \{+\infty\}$ is defined by $\mu(s) = \inf\{k \in \mathbb{N} : s \models \forall \square(\text{wgt} > -k)\}$.

Proposition 1. $\Pr(\square(\text{wgt} > z)) > 0$ iff there exists a BSCC \mathcal{C} with $s \in \mathcal{C}$ such that one of the following two conditions holds:

- (i) $\mathbb{E}_{\mathcal{C}}(\text{wgt}) > 0$ and $cdist(s, z) = +\infty$, or
- (ii) $\mathbb{E}_{\mathcal{C}}(\text{wgt}) = 0$, \mathcal{C} contains no negative cycles, and $\mu(s) + z < cdist(s, z)$.

Our proof of Proposition 1 shares some ideas with proofs presented in [10,12] for unit-weighted Markov chains and $\nabla \in \{\diamond, \square\}$.

Intuitively, (i) requires that there exists a path $\hat{\pi}$ reaching \mathcal{C} which contains a positive cycle and where the accumulated weight of each prefix is always greater than z . As $\mathbb{E}_{\mathcal{C}}(\text{wgt}) > 0$ ensures that for every $s \in \mathcal{C}$ there exists $n \in \mathbb{N}$ with $\Pr_s^{\mathcal{M}}(\square(\text{wgt} > -n)) > 0$, the existence of $\hat{\pi}$ implies $\Pr(\square(\text{wgt} > z) \wedge \diamond\mathcal{C}) > 0$. Condition (ii) asks for a path $\hat{\pi}$ reaching a state $s \in \mathcal{C}$ accumulating at least $\mu(s) + z$, where the accumulated weight of each non-empty prefix always exceeds z . As a BSCC with expected weight zero contains a negative cycle if and only if it contains a positive cycle, $\mu(s)$ is finite and $\Pr_s^{\mathcal{M}}(\square(\text{wgt} > -\mu(s))) = 1$. Hence, the existence of such $\hat{\pi}$ also implies $\Pr(\square(\text{wgt} > z) \wedge \diamond\mathcal{C}) > 0$.

Concerning the reverse implication, i.e., $\Pr(\square(\text{wgt} > z)) > 0$ implies either condition (i) or (ii), notice that $\Pr(\square(\text{wgt} > z)) > 0$ entails the existence of a BSCC \mathcal{C} such that $\Pr(\square(\text{wgt} > z) \wedge \diamond\mathcal{C}) > 0$. If $\mathbb{E}_{\mathcal{C}}(\text{wgt}) > 0$, then there is a finite path starting in ι whose weight does not drop below z and which ends in a positive cycle from some $s \in \mathcal{C}$. This fact finally implies (i). If \mathcal{C} is as in (ii), we rely on the observation that then $\Pr_s^{\mathcal{M}}(\square(\text{wgt} > -\mu(s)+1)) = 0$ for all $s \in \mathcal{C}$. For all other BSCCs \mathcal{C} we have $\Pr(\square(\text{wgt} > z) \wedge \diamond\mathcal{C}) = 0$.

Analogous characterizations can be provided for the constraints $\Pr(\square\diamond(\text{wgt} > z)) > 0$ and $\Pr(\square\diamond(\text{wgt} > z)) = 1$, in which cases it suffices to consider standard shortest or longest paths instead of the constraint distance.

Polynomial-Time Decision Procedure. Due to Proposition 1 it suffices to argue that $cdist(s, z)$ and $\mu(s)$ can be computed in polynomial time in order to complete the proof of Theorem 1. Whereas $cdist(s, z)$ can be computed using a modified Bellman-Ford algorithm, $\mu(s)$ is computable using any standard shortest-path algorithm. For every BSCC \mathcal{C} as in (ii) holds $\mu(s) - 1 = -\min_{s' \in \mathcal{C}} dist_{\min}(s, s')$, where $dist_{\min}(s, s')$ is the length of the shortest path from s to s' . Remember, if \mathcal{C} is a BSCC with $\mathbb{E}_{\mathcal{C}}(\text{wgt}) = 0$ and \mathcal{C} contains no negative cycles, then the accumulated weight of each cycle in \mathcal{C} is zero. Notice, $\mu(s) \in \mathbb{N}$ for all $s \in \mathcal{C}$.

Remark 1. In order to decide the qualitative decision problems of Theorem 2 without side constraints φ , it suffices to consider the underlying graph structure of the Markov chain and the expected weights of the BSCCs. Hence, using standard approaches we can generalize the decision procedure to involve non-trivial

ω -regular side constraints. If φ is a Streett or Rabin condition, then the BSCCs of \mathcal{M} can be partitioned into sets \mathcal{C}^φ and $\mathcal{C}^{\neg\varphi}$ such that for all infinite paths π eventually entering some BSCC \mathcal{C} , $\pi \models \varphi$ if and only if $\mathcal{C} \in \mathcal{C}^\varphi$.

A natural question is whether the result of Theorem 2 can be extended when dealing with constraints for multiple weight functions. The closely related termination problem for Markov chains with multiple unit-weight functions is known to be solvable in polynomial time in the size of the Markov chain and in exponential time in the number of weight functions [12]. However, by exploiting the ExpSpace-completeness of the coverability problem for vector addition systems with states [8, 13, 29], we obtain the following result.

Proposition 2. *Given weight functions $\text{wgt}_1, \dots, \text{wgt}_d$ and $z_1, \dots, z_d \in \mathbb{Z}$, the problem whether $\Pr(\Box(\text{wgt}_1 > z_1) \wedge \dots \wedge \Box(\text{wgt}_d > z_d)) > 0$ is ExpSpace-complete.*

4 Computing Ratio and Weight Quantiles

In this section, we sketch the proof of our main result (Theorem 1) stating that ratio and weight quantiles are computable in polynomial time. As discussed in Remark 1, it suffices to consider quantiles without side constraints, i.e., $\varphi = \text{true}$. In the remainder of this section let \mathcal{M} be either a weighted Markov chain or an energy-utility Markov chain as introduced in Sect. 2. Assume ∇ is given as before. We use $\text{Qu}^*[\nabla\text{wgt}]$ as a short-hand notation for both $\text{Qu}_{\text{true}}^{>0}[\nabla\text{wgt}]$ and $\text{Qu}_{\text{true}}^{=1}[\nabla\text{wgt}]$. $\text{Qu}^*[\nabla\text{ratio}]$ is used analogously.

Whereas the weight quantile values $\text{Qu}^*[\nabla\text{wgt}]$ can either be computed directly using a BSCC analysis or a binary search on a finite interval of integers, the ratio quantiles are much more involved. The standard reduction of ratio constraints to weight constraints used in Sect. 3 is not adequate for computing ratio quantiles. This is mainly due to fact that the transformation from ratio objectives ($\text{ratio} > q$) to weight objectives ($\text{wgt}_q > 0$) depends on the threshold q . We even have to show that ratio quantiles are rational as the supremum of an infinite subset of \mathbb{Q} can be irrational. Our computation scheme for ratio quantiles relies on the identification of a finite, but exponentially large set of rational values containing the quantile value, and on the reduction to a best-approximation problem, which can be solved in polynomial time using the continued-fraction method.

4.1 Weight Quantiles

We sketch the approach towards polynomial-time algorithms for the computation of ∇ -weight quantiles by considering quantiles of the form $\text{Qu}^*[\Box\text{wgt}]$. The argument for $\text{Qu}^*[\Diamond\text{wgt}]$ is analogous and the quantiles $\text{Qu}^*[\Diamond\Box\text{wgt}]$ and $\text{Qu}^*[\Box\Diamond\text{wgt}]$ can be computed by using BSCC analysis only.

The almost-sure \Box -weight quantile $\text{Qu}^{=1}[\Box\text{wgt}]$ can be computed using standard minimal-distance algorithms, as for every $z \in \mathbb{Z}$, $\Pr(\Box(\text{wgt} > z)) = 1$ if and only if there is no path $\hat{\pi} \in \text{FinPaths}_l^{\mathcal{M}}$ satisfying $\text{wgt}(\hat{\pi}) \leq z$ [3, 6]. In order to compute the positive \Box -weight quantile $\text{Qu}^{>0}[\Box\text{wgt}]$ it suffices to compute

integers a and b such that $\text{Qu}^{>0}[\square\text{wgt}] \in ([a, b] \cap \mathbb{Z}) \cup \{-\infty\}$. If a and b are computable in polynomial time, $\text{Qu}^{>0}[\square\text{wgt}]$ is computable in polynomial time using a binary search on $[a, b] \cap \mathbb{Z}$ together with the decision procedure of Theorem 2.

Proposition 3. *Let \min_{wgt} and \max_{wgt} be the minimum and the maximum of wgt, respectively. Then,*

$$\text{Qu}^{>0}[\square\text{wgt}] \in \{z \in \mathbb{Z} : \min\{0, 2 \cdot |S| \cdot \min_{\text{wgt}}\} \leq z \leq \max_{\text{wgt}}\} \cup \{-\infty\}.$$

4.2 Ratio Quantiles

The computation of ∇ -ratio quantiles relies on the idea that one can effectively approximate the quantile value up to an arbitrarily precision using Theorem 2. We can then deduce the exact quantile value from a sufficiently accurate approximation by solving a best-approximation problem using the continued-fraction method. Crucial for applying this technique is that $\text{Qu}^*[\nabla\text{ratio}]$ turns out to be rational and its denominator is bounded by some computable natural number. These ideas are summarized in the following lemma, which is proven by solving the following *best-approximation problem* [22]: Given $N \in \mathbb{N}_{>0}$ and $\alpha \in \mathbb{Q}$, one asks for some $\beta \in \mathbb{Q}$ with denominator at most N that minimizes $|\alpha - \beta|$. By [22, Theorem 5.1.9], this best-approximation problem for N and α is solvable in time polynomial in the encoding length of N and α , using the so-called *continued-fraction method*.

Lemma 1. *For $N \in \mathbb{N}$ and $\alpha \in \mathbb{Q}_{\geq 0}$ there exists at most one $\beta \in \mathbb{Q}$ such that $|\alpha - \beta| < 1/(2N^2)$ and $\beta \in \{a/b : a \in \mathbb{N} \text{ and } b \in [1, N] \cap \mathbb{N}\}$. If such a β exists, it can be computed in time polynomial in the encoding length of N and α .*

In fact, if such a β exists, the requirement $|\alpha - \beta| < 1/(2N^2)$ ensures that it is the unique solution of the best-approximation problem for N and α . We now focus on relating ratio quantiles to Lemma 1, i.e., we show that we can find $N \in \mathbb{N}$ and $\alpha \in \mathbb{Q}$ such that $|\alpha - \text{Qu}^*[\nabla\text{ratio}]| < 1/(2N^2)$ as well as $\text{Qu}^*[\nabla\text{ratio}] \in \{a/b : a \in \mathbb{N} \text{ and } b \in [1, N] \cap \mathbb{N}\}$. Notice that if both, N and α are computable in polynomial time, we can conclude Theorem 1 for the ratio case. Intuitively spoken, the first requirement asks for an approximation of the quantile up to a given precision. The second one demands that the ratio quantile is rational and that its denominator can be bounded.

Approximation of Ratio Quantiles. We can check in polynomial time whether $\text{Pr}(\nabla(\text{ratio} > q)) > 0$ holds for a given rational number q (Theorem 2). The same applies for the almost-sure case. Hence, given $r \in \mathbb{Q}$ such that $\text{Qu}^*[\nabla\text{ratio}] \leq r$, we can approximate the quantile up to an arbitrary precision ε in time polynomial in the size of \mathcal{M} and logarithmic in $1/\varepsilon$ and r using a binary search on the interval $[0, r]$. When \max_u denotes the maximum of utility and \min_e the minimum of energy, $\text{Qu}^*[\nabla\text{ratio}] \leq r$ with $r = \max_u/\min_e$, since every finite path $\hat{\pi}$ satisfies $\text{ratio}(\hat{\pi}) \leq \max_u/\min_e$. Remember that the requirements of energy ensure $\min_e \neq 0$. Clearly, \max_u and \min_e are computable in polynomial time in the size of energy and utility. These facts yield to the following proposition.

Proposition 4. For $N \in \mathbb{N}$ there is an $\alpha \in \mathbb{Q}$ computable in polynomial time such that $|\alpha - \text{Qu}^*[\nabla\text{ratio}]| \leq 1/(2N^2)$.

Bounding the Denominator. It is not enough to show that the quantile $\text{Qu}^*[\nabla\text{ratio}]$ is rational in order to apply Lemma 1. We have to furthermore show that the denominator of $\text{Qu}^*[\nabla\text{ratio}]$ can be bounded and such a bound can be computed in polynomial time. The idea is to state finitely many rational numbers which serve as candidates for the exact quantile values. Let

$$\begin{aligned} Q_{BSCC} &= \{\mathbb{L}_{\mathcal{C}}(\text{ratio}) : \mathcal{C} \text{ is a BSCC}\}, \\ Q_{\text{path}} &= \{\text{ratio}(\hat{\pi}) : \hat{\pi} \text{ simple path starting in } \iota\}, \text{ and} \\ Q_{\text{cycle}} &= \{\text{ratio}(\vartheta) : \vartheta \text{ simple cycle}\}. \end{aligned}$$

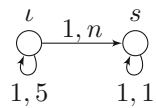
Each of these sets are finite, contain only rational values and may provide candidates for the quantile values $\text{Qu}^*[\square\text{ratio}]$. In fact, all possible candidates for $\text{Qu}^*[\nabla\text{ratio}]$ are covered through $Q_{\text{all}} = Q_{BSCC} \cup Q_{\text{path}} \cup Q_{\text{cycle}}$.

Proposition 5. $\text{Qu}^*[\nabla\text{ratio}] \in Q_{\text{all}} = Q_{BSCC} \cup Q_{\text{path}} \cup Q_{\text{cycle}}$.

We now sketch a proof for Proposition 5. If $\nabla \in \{\diamond\square, \square\diamond\}$ the claim is a consequence of the fact that almost all paths $\pi \in \text{InfPaths}_i^M$ eventually enter a BSCC \mathcal{C} and $\lim_{n \rightarrow \infty} \text{ratio}(\pi[. . . n]) = \mathbb{L}_{\mathcal{C}}(\text{ratio})$. Hence, we even have the stronger result $\text{Qu}^*[\diamond\square\text{ratio}], \text{Qu}^*[\square\diamond\text{ratio}] \in Q_{BSCC}$. Thus, for these quantiles we establish a polynomial-time computation scheme without employing Lemma 1.

To prove $\text{Qu}^{>0}[\square\text{ratio}] \in Q_{\text{all}}$ we show the following statement: If $q \in \mathbb{Q} \setminus Q_{\text{all}}$ and $\Pr(\square(\text{ratio} > q)) > 0$, then there exists $q' \in Q_{\text{all}}$, $q < q'$ such that for all $\varepsilon \in \mathbb{Q}_{>0}$, $\Pr(\square(\text{ratio} > q' - \varepsilon)) > 0$. Assume $\Pr(\square(\text{ratio} > q)) > 0$ where $q \in \mathbb{Q} \setminus Q_{\text{all}}$. Using the assumption $q \notin Q_{\text{all}}$, Proposition 1 together with the ratio-weight transformation (c.f. Sect. 3) implies the existence of a BSCC \mathcal{C} and a finite path $\hat{\pi}_q$ reaching \mathcal{C} such that $\mathbb{L}_{\mathcal{C}}(\text{ratio}) > q$, $\hat{\pi}_q$ contains a cycle ϑ with $\text{ratio}(\vartheta) > q$, and every non-empty prefix ρ of $\hat{\pi}_q$ satisfies $\text{ratio}(\rho) > q$. We call $\hat{\pi}_q$ a *witness* for $\Pr(\square(\text{ratio} > q)) > 0$. Let $q' \in Q_{\text{all}}$ be the smallest element of Q_{all} such that $q < q'$ and $\varepsilon \in \mathbb{Q}_{>0}$. Outgoing from $\hat{\pi}_q$ we can construct a finite path $\hat{\pi}'_{q' - \varepsilon}$ that serves as a witness for $\Pr(\square(\text{ratio} > q' - \varepsilon)) > 0$.

For example, consider the Markov chain with the two states ι and s depicted on the right, depending on $n \in \mathbb{N}$. First suppose $n = 100$. Using the characterization given in Proposition 1 and the ratio-weight transformation, $\hat{\pi}_{2/100} = \iota \iota \iota s$ is a witness for $\Pr(\square(\text{ratio} > 2/100)) > 0$. Since we can iterate ι arbitrarily often, given $\varepsilon \in \mathbb{Q}_{>0}$ there exists $k \in \mathbb{N}$ such that $\hat{\pi}_{1/5 - \varepsilon} = \iota^k s$ is a witness for $\Pr(\square(\text{ratio} > 1/5 - \varepsilon)) > 0$. If we assume $n = 1$, then $\hat{\pi}'_{1/6} = \iota \iota \iota s$ is a witness for $\Pr(\square(\text{ratio} > 1/6)) > 0$. As $\text{ratio}(\iota \iota) < \text{ratio}(\hat{\pi}'_{1/6})$ we now do not iterate the state ι and achieve a witness for $(1 - \varepsilon)$. For arbitrary $\varepsilon \in \mathbb{Q}_{>0}$ the path $\hat{\pi}'_{1 - \varepsilon} = \iota s$ is a witness for $\Pr(\square(\text{ratio} > 1 - \varepsilon)) > 0$.



Energy-utility Markov chain. Transitions are labeled by utility, energy.

Remark 2. In fact one can improve Proposition 5:

$$\begin{aligned} \text{Qu}^{\leq 1}[\Box\text{ratio}] &= \min(Q_{\text{cycle}} \cup Q_{\text{path}}), \\ \text{Qu}^{> 0}[\Diamond\text{ratio}] &= \max(Q_{\text{path}} \cup Q_{\text{cycle}}), \\ \text{Qu}^{\leq 1}[\Box\Diamond\text{ratio}] &= \text{Qu}^{\leq 1}[\Diamond\Box\text{ratio}] = \min(Q_{\text{BSCC}}), \text{ and} \\ \text{Qu}^{> 0}[\Box\Diamond\text{ratio}] &= \text{Qu}^{> 0}[\Diamond\Box\text{ratio}] = \max(Q_{\text{BSCC}}). \end{aligned}$$

Let us now draw the consequences towards a proof of Theorem 1. Since Q_{all} is exponential, Proposition 5 directly yields an exponential-time computation procedure for ratio quantiles. However, we can avoid the computation of Q_{all} : Proposition 5 implies that the denominator of a ratio quantile is bounded by the maximum of the set $M = \{|S| \cdot \max_e\} \cup \{\text{den}(\mathbb{L}_{\mathcal{C}}(\text{ratio})) : \mathcal{C} \text{ is a BSCC}\}$. Here, \max_e is the maximum of the function energy, and $\text{den}(q)$ denotes the denominator of a rational number q . The set M can be computed in polynomial time and hence we obtain the following proposition, which completes our proof for Theorem 1.

Proposition 6. *One can compute a natural number N in polynomial time such that $\text{Qu}^*[\nabla\text{ratio}] \in \{a/b : a \in \mathbb{N} \text{ and } b \in [1, N] \cap \mathbb{N}\}$.*

Putting things together, we obtain the open part of Theorem 1, i.e., the quantile $\text{Qu}^*[\nabla\text{ratio}]$ can be computed in polynomial time: First compute N as given in Proposition 6, then compute α from Proposition 4, and finally apply Lemma 1 and solve the best-approximation problem for N and α .

Remark 3. Mean-payoff objectives [9,30] are tightly connected to our $\Diamond\Box$ -ratio objectives where energy is restricted to $\text{energy}(s, s') = 1$ for all $s, s' \in S$. When $\pi = s_0 s_1 s_2 \dots$ is an infinite path, its *mean-payoff* is defined by

$$\text{MP}(\pi) = \liminf_{n \rightarrow \infty} \frac{1}{n+1} \sum_{i=0}^n \text{utility}(s_i, s_{i+1}).$$

Then, e.g., $\text{Qu}^{> 0}[\Diamond\Box\text{ratio}] = \sup\{q \in \mathbb{Q} : \Pr(\text{MP} > q) > 0\}$, which is a consequence of the following implications with $\pi \in \text{InfPaths}^{\mathcal{M}}$ and $q \in \mathbb{Q}_{\geq 0}$: $\pi \models \Diamond\Box(\text{ratio} > q)$ implies $\text{MP}(\pi) \geq q$, and vice versa, $\text{MP}(\pi) > q$ implies $\pi \models \Diamond\Box(\text{ratio} \geq q)$. Notice that [9] and [30] consider MDPs instead of Markov chains.

5 Conclusions

We considered the optimization problem of computing quantile values over a simple parametrized logic with ratio and weight objectives. For weighted Markov chains we established efficient algorithms to exactly compute quantiles where the ratio or weight objective under an ω -regular side constraint has to be fulfilled almost-surely or with positive probability. Whereas in the case of weight quantiles a simple binary search and a polynomial-time decision procedure for weight objectives is sufficient, ratio quantiles required a more sophisticated approach, namely by applying the continued-fraction method. The presented techniques can be used to establish polynomial-time algorithms also for various extensions of the objectives we considered. For instance, the initialization phase after which the

ratio threshold q has to be always exceeded in a persistence event $\diamond\Box(\text{ratio}>q)$ might be conditioned by a set of triggering states T , which could be expressed in LTL fashion as an event $(\neg T)\mathcal{U}(T \wedge \Box(\text{ratio}>q))$ [31].

The extension towards p -quantiles, i.e., quantiles where the ratio or weight objective holds not only almost surely or with positive probability but its probability exceeds a given threshold p , is not straight forward and an efficient exact computation cannot be expected. Already the corresponding \Box -weight decision problem in unit-weight Markov chains can be shown to be POSSLP-hard [3, 10, 19]. Thus, a polynomial-time (or even NP) decision procedure is not possible without a major breakthrough in exact numerical analysis. However, weight p -quantiles can be computed by computing corresponding almost-sure and positive quantiles $q_{=1}$ and $q_{>0}$ in polynomial time as presented in this paper, and performing a simple binary search on $[q_{=1}, q_{>0}]$ applying well-known polynomial-space algorithms for analyzing probabilistic one-counter automata [11]. As weight and ratio decision problems are interreducible in linear time, this approach can also be used to approximate ratio p -quantiles.

Acknowledgements. We thank Stefan Kiefer for pointing us to the continued-fraction method and its application [20].

References

1. Andova, S., Hermanns, H., Katoen, J.P.: Discrete-time rewards model-checked. In: Larsen, K.G., Niebert, P. (eds.) FORMATS 2003. LNCS, vol. 2791, pp. 88–104. Springer, Heidelberg (2004)
2. Baier, C., Daum, M., Dubslaff, C., Klein, J., Klüppelholz, S.: Energy-utility quantiles. In: Badger, J.M., Rozier, K.Y. (eds.) NFM 2014. LNCS, vol. 8430, pp. 285–299. Springer, Heidelberg (2014)
3. Baier, C., Dubslaff, C., Klein, J., Klüppelholz, S., Wunderlich, S.: Probabilistic model checking for energy-utility analysis. In: van Breugel, F., Kashefi, E., Palamidessi, C., Rutten, J. (eds.) Horizons of the Mind. LNCS, vol. 8464, pp. 96–123. Springer, Heidelberg (2014)
4. Baier, C., Dubslaff, C., Klüppelholz, S.: Trade-off analysis meets probabilistic model checking. In: CSL-LICS 2014, pp. 1:1–1:10. ACM (2014)
5. Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press, Cambridge (2008)
6. Baier, C., Klein, J., Klüppelholz, S., Wunderlich, S.: Weight monitoring with linear temporal logic: complexity and decidability. In: CSL-LICS 2014, pp. 11:1–11:10. ACM (2014)
7. Boker, U., Chatterjee, K., Henzinger, T.A., Kupferman, O.: Temporal specifications with accumulative values. In: LICS 2011, pp. 43–52. IEEE Computer Society (2011)
8. Bozzelli, L., Ganty, P.: Complexity analysis of the backward coverability algorithm for VASS. In: Delzanno, G., Potapov, I. (eds.) RP 2011. LNCS, vol. 6945, pp. 96–109. Springer, Heidelberg (2011)

9. Brázdil, T., Brozek, V., Chatterjee, K., Forejt, V., Kucera, A.: Two views on multiple mean-payoff objectives in Markov decision processes. *Logical Methods Comput. Sci.* **10**(1), 1–29 (2014)
10. Brázdil, T., Brozek, V., Etessami, K., Kucera, A., Wojtczak, D.: One-counter Markov decision processes. In: *SODA 2010*, pp. 863–874. SIAM (2010)
11. Brázdil, T., Esparza, J., Kiefer, S., Kucera, A.: Analyzing probabilistic pushdown automata. *Formal Methods Syst. Des.* **43**(2), 124–163 (2013)
12. Brázdil, T., Kiefer, S., Kucera, A., Novotný, P., Katoen, J.-P.: Zero-reachability in probabilistic multi-counter automata. In: *CSL-LICS 2014*. ACM (2014)
13. Cardoza, E., Lipton, R., Meyer, A.R.: Exponential space complete problems for Petri nets and commutative semigroups (preliminary report). In: *STOC 1976*, pp. 50–54. ACM (1976)
14. Chatterjee, K., Doyen, L.: Energy and mean-payoff parity Markov decision processes. In: Murlak, F., Sankowski, P. (eds.) *MFCS 2011*. LNCS, vol. 6907, pp. 206–218. Springer, Heidelberg (2011)
15. Chatterjee, K., Doyen, L.: Energy parity games. *Theoret. Comput. Sci.* **458**, 49–60 (2012)
16. Clarke, E., Grumberg, O., Peled, D.: *Model Checking*. MIT Press, Cambridge (2000)
17. de Alfaro, L.: *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, Department of Computer Science (1997)
18. Etessami, K., Kwiatkowska, M., Vardi, M.Y., Yannakakis, M.: Multi-objective model checking of Markov decision processes. *Logical Methods Comput. Sci.* **4**(4), 1–21 (2008)
19. Etessami, K., Yannakakis, M.: Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM* **56**(1), 1:1–1:66 (2009)
20. Etessami, K., Yannakakis, M.: On the complexity of Nash equilibria and other fixed points. *SIAM J. Comput.* **39**(6), 2531–2597 (2010)
21. Freedman, D.: *Markov Chains*. Springer, New York (1983)
22. Grötschel, M., Lovász, L., Schrijver, A.: *Geometric Algorithms and Combinatorial Optimization*. Springer, Heidelberg (1993)
23. Haase, C., Kiefer, S.: The odds of staying on budget. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) *ICALP 2015*. LNCS, vol. 9135, pp. 234–246. Springer, Heidelberg (2015)
24. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: a tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) *TACAS 2006*. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
25. Juhl, L., Gulstrand Larsen, K., Raskin, J.-F.: Optimal bounds for multiweighted and parametrised energy games. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) *He Festschrift*. LNCS, vol. 8051, pp. 244–255. Springer, Heidelberg (2013)
26. Kallenberg, O.: *Foundations of Modern Probability*. Springer, New York (2002)
27. Katoen, J.-P., Zapreev, I., Hahn, E., Hermanns, H., Jansen, D.: The ins and outs of the probabilistic model checker MRMC. *Perform. Eval.* **68**(2), 90–104 (2011)
28. Krähmann, D., Schubert, J., Baier, C., Dubslaff, C.: Ratio and weight quantiles. Technical report, Technische Universität Dresden (2015). <http://wwwwtcs.inf.tu-dresden.de/ALGI/PUB/MFCS15/>
29. Rackoff, C.: The covering and boundedness problems for vector addition systems. *Theoret. Comput. Sci.* **6**(2), 223–231 (1978)
30. Randour, M., Raskin, J.-F., Sankur, O.: Percentile queries in multi-dimensional Markov decision processes. In: *CAV 2015*. LNCS. Springer, Heidelberg (2015, to appear)

31. Schubert, J.: Weight and ratio objectives in annotated Markov chains. Master's thesis, TU Dresden (2015)
32. Ummels, M., Baier, C.: Computing quantiles in Markov reward models. In: Pfening, F. (ed.) FOSSACS 2013 (ETAPS 2013). LNCS, vol. 7794, pp. 353–368. Springer, Heidelberg (2013)
33. von Essen, C., Jobstmann, B.: Synthesizing systems with optimal average-case behavior for ratio objectives. In: iWIGP 2011. EPTCS, vol. 50, pp. 17–32 (2011)

Precise Upper and Lower Bounds for the Monotone Constraint Satisfaction Problem

Victor Lagerkvist^(✉)

Department of Computer and Information Science,
Linköping University, Linköping, Sweden
victor.lagerkvist@liu.se

Abstract. The *monotone constraint satisfaction problem* (MCSP) is the problem of, given an existentially quantified positive formula, decide whether this formula has a model. This problem is a natural generalization of the constraint satisfaction problem, which can be seen as the problem of determining whether a conjunctive formula has a model. In this paper we study the worst-case time complexity, measured with respect to the number of variables, n , of the MCSP problem parameterized by a constraint language Γ (MCSP(Γ)). We prove that the complexity of the NP-complete MCSP(Γ) problems on a given finite domain D falls into exactly $|D| - 1$ cases and ranges from $O(2^n)$ to $O(|D|^n)$. We give strong lower bounds and prove that MCSP(Γ), for any constraint language Γ over any finite domain, is solvable in $O(|D'|^n)$ time, where D' is the domain of the core of Γ , but not solvable in $O(|D'|^{\delta n})$ time for any $\delta < 1$, unless the strong exponential-time hypothesis fails. Hence, we obtain a complete understanding of the worst-case time complexity of MCSP(Γ) for constraint languages over arbitrary finite domains.

1 Introduction

The *constraint satisfaction problem* over a constraint language Γ (CSP(Γ)) is a widely studied computational problem which can be described as the problem of, given a conjunctive formula over Γ , verify whether there exists a model of this formula. In general the CSP(Γ) problem is NP-complete, and much research has been made to separate tractable from NP-hard cases [3, 13]. A related question to establishing dichotomies between tractable and intractable cases is to study the complexity differences between NP-complete CSP problems. Let n denote the number of variables in a given CSP(Γ) instance. Is it then, for example, possible to characterize all constraint languages Γ such that CSP(Γ) is solvable in $O(c^n)$ time for some $c \in \mathbb{R}$? Ultimately, one would like to have a table, which for every constraint language Γ contains a constant $c \in \mathbb{R}$ such that CSP(Γ) is solvable in $O(c^n)$ time but not in $O(d^n)$ time for any $d < c$. Clearly, even assuming $P \neq NP$, such a table would be very difficult, if not impossible, to obtain. A more feasible approach is to order all NP-complete CSP problems by their relative worst-case time complexity, i.e., CSP(Γ) lies below CSP(Δ) in this

ordering if $\text{CSP}(\Gamma)$ is solvable in $O(c^n)$ time whenever $\text{CSP}(\Delta)$ is solvable in $O(c^n)$ time. Jonsson et al. [9] studied the structure of this ordering for Boolean CSP problems and proved that it had a minimal element. With this “easiest” Boolean CSP problem they obtained lower bounds for all NP-complete Boolean CSP problems and proved that there does not exist any NP-complete Boolean CSP(Γ) problem solvable in subexponential time, unless the exponential-time hypothesis fails. A similar study was later conducted for Boolean optimization problems where analogous results and lower bounds were obtained [10].

In this paper we continue this line of research in the context of the *monotone constraint satisfaction problem* parameterized by a constraint language Γ ($\text{MCSP}(\Gamma)$). This problem can be viewed as a generalization of the CSP problem where the objective is to determine whether an existentially quantified first-order formula without negation over Γ has a model. This problem has been studied by Hermann and Richoux [6], who gave a dichotomy theorem for arbitrary finite domains, separating tractable from NP-complete cases. This result was later extended to also cover the case of infinite domains [1]. Closure properties of disjunctive logic formulas was investigated by Fargier and Marquis [5] but with regards to knowledge representation problems. We are now interested in the aforementioned questions regarding the worst-case time complexity of $\text{MCSP}(\Gamma)$, and in particular how well the ordering between the complexity of NP-complete $\text{MCSP}(\Gamma)$ problems can be approximated. As a tool to compare and relate worst-case running times between NP-complete $\text{MCSP}(\Gamma)$ problems, we utilize a restricted form of polynomial-time reductions, which only increases the number of variables within instances by a constant, *constant variable reductions* (CV-reductions). It is readily verified that if $\text{MCSP}(\Gamma)$ is CV-reducible to $\text{MCSP}(\Delta)$ then $\text{MCSP}(\Gamma)$ is solvable in $O(c^n)$ time whenever $\text{MCSP}(\Delta)$ is solvable in $O(c^n)$ time (where n denotes the number of variables in a given instance). We begin in Sect. 3.1 by first giving a straightforward condition to check whether $\text{MCSP}(\Gamma)$ is CV-reducible to $\text{MCSP}(\Delta)$. This proof makes use of the *algebraic approach* to constraint satisfaction problems [8] and the Galois connection between strong partial endomorphism monoids and weak Krasner algebras without existential quantification [2]. In Sect. 3.2 we use this result to obtain a full understanding of the applicability of CV-reductions to the $\text{MCSP}(\Gamma)$ problem. Let Γ be an arbitrary constraint language over a finite domain D and let D' denote the domain of the core of Γ (the reader unfamiliar with this model theoretical concept is advised to quickly consult Sect. 2.2). We prove that $\text{MCSP}(\Gamma)$ is exactly as hard as the full MCSP over D' , i.e. the MCSP problem where all relations over D' are allowed to occur in constraints. This furthermore implies that there are at most $|D| - 1$ possible cases for the worst-case complexity of $\text{MCSP}(\Gamma)$.

With the help of the results from Sect. 3 we in Sect. 4 turn to the problem of determining lower bounds for $\text{MCSP}(\Gamma)$. To prove these lower bounds we relate the complexity of MCSP to the *strong exponential-time hypothesis* (SETH), i.e. the conjecture that the Boolean satisfiability problem is not solvable in $O(2^{\delta n})$ time for any $\delta < 1$ [4, 7]. We prove the following: if the SETH

holds then $\text{MCSP}(\Gamma)$ is solvable in $O(c^n)$ time but not in $O(c^{\delta n})$ time for any $\delta < 1$, where c is the size of the domain of the core of Γ . As a side result we prove an analogous result also hold for the CSP problem: if the Boolean CSP problem is not solvable in $O(2^{\delta n})$ for any $\delta < 1$ then the CSP problem over any finite domain D is not solvable in $O(|D|^{\delta n})$ for any $\delta < 1$. Hence, for any finite domain D and any $c \in \{1, \dots, |D|\}$, we obtain a complete classification of the MCSP problems solvable in $O(c^n)$ time but not in $O(c^{\delta n})$ time for any $\delta < 1$. In contrast to the CSP problem we can therefore not only approximate the ordering between the complexity of NP-complete MCSP problems, but actually obtain a complete understanding. While these results do not directly carry over to the CSP problem, we still believe that some of the involved techniques could be useful when studying the time complexity of other problems parameterized by constraint languages.

2 Preliminaries

In this section we introduce constraint languages, the monotone constraint satisfaction problem, and give a brief introduction to the necessary algebraic concepts required in the subsequent treatment.

2.1 Constraint Languages and Functions

Let $D \subset \mathbb{N}$ be a finite domain. Let Rel_D denote the set of all finitary relations over D . For a relation $R \in \text{Rel}_D$ we let $\text{ar}(R)$ denote its arity. For each $i \in D$ let c_i denote the constant relation $\{(i)\}$. Let $\text{id}_D(i) = i$ be the identity function over D , $\text{Eq}_D = \{(i, i) \mid i \in D\}$ the equality relation, and $\text{Neq}_D = \{(i, j) \in D^2 \mid i \neq j\}$ the inequality relation. Given a function f on D we let $\text{img}(f)$ denote its image. If the domain is clear from the context we simply write id , Eq and Neq , respectively. A *constraint language* Γ over D is a finite or infinite set of relations $\Gamma \subseteq \text{Rel}_D$ such that $\text{Eq}_D \in \Gamma$. Hence, whenever we speak of a constraint language we assume that this language contains the equality relation.

We usually represent relations as logical formulas, and use the notation $R(x_1, \dots, x_n) \equiv \phi$ to denote the n -ary relation $R = \{(f(x_1), \dots, f(x_n)) \mid f \text{ is a model of } \phi\}$. A *monotone formula* over a constraint language Γ is logical formula with free variables x_1, \dots, x_n of the form $\exists y_1, \dots, y_m. \phi$, where ϕ is a quantifier-free formula over $x_1, \dots, x_n, y_1, \dots, y_m$ consisting of disjunction, conjunction, over positive literals of the form $R_i(\mathbf{x}_i)$, where $R_i \in \Gamma$, and \mathbf{x}_i is a tuple of variables over $x_1, \dots, x_n, y_1, \dots, y_m$. A *quantifier-free monotone formula* is a monotone formula without existential quantification. These restricted classes of logical formulas will be important in Sect. 2.3 when we define the monotone constraint satisfaction problem, and in Sects. 2.4 and 2.5 where we discuss closure properties of relations.

2.2 Cores of Constraint Languages

A constraint language $\{R'_1, \dots, R'_m\}$ over $D' \subseteq D$ is a *substructure* of a constraint language $\{R_1, \dots, R_m\}$ over D if each $R'_i \subseteq R_i$. A *homomorphism* h of a constraint language $\{R_1, \dots, R_m\}$ to a constraint language $\{R'_1, \dots, R'_m\}$ over D' is a function $h : D \rightarrow D'$ such that $(h(a_1), \dots, h(a_{\text{ar}(R_i)})) \in R'_i$ for every i and every $(a_1, \dots, a_{\text{ar}(R_i)}) \in R_i$. Here we tacitly assume that constraint languages can be viewed as relational structures, i.e. that the relations are given as an ordered sequence and have an associated signature. A constraint language Γ is a *core* if there does not exist a homomorphism to a proper substructure of Γ , and we say that Γ' is a core of Γ if Γ' is a core and there exists a homomorphism from Γ to Γ' . Since the cores of a constraint language Γ are equivalent up to isomorphism we typically speak of *the core* of Γ and let $\text{Core}(\Gamma)$ denote this constraint language. The *core-size* of Γ is the size of the domain of $\text{Core}(\Gamma)$.

2.3 The Monotone Constraint Satisfaction Problem

Let Γ be a constraint language over D . Recall that a constraint language in our notation always includes the equality relation over D . The *monotone constraint satisfaction problem* over Γ ($\text{MCSP}(\Gamma)$) is defined as follows.

INSTANCE: A tuple (V, ϕ) where V is a set of variables and ϕ a quantifier-free monotone formula over Γ and V .

QUESTION: Is there a function $f : V \rightarrow D$ such that f is a model of ϕ ?

Even if $\text{Eq} \notin \Gamma$ we typically write $\text{MCSP}(\Gamma)$ instead of $\text{MCSP}(\Gamma \cup \{\text{Eq}\})$. Given an instance $I = (V, \phi)$ of $\text{MCSP}(\Gamma)$ we let $\|I\|$ denote the number of bits required to represent I and $\text{Constraints}(I) = \{R_i(\mathbf{x}_i) \mid R_i(\mathbf{x}_i) \text{ is a constraint application in } \phi\}$, where each $R_i \in \Gamma$ and \mathbf{x}_i is a tuple of variables over V of length $\text{ar}(R_i)$.

Example 1. Consider the problem $\text{MCSP}(\{c_1, c_2\})$ over the Boolean domain $\{1, 2\}$, where $c_1 = \{(1)\}$ and $c_2 = \{(2)\}$. Then $\text{MCSP}(\{c_1, c_2\})$ can be seen as a variant of the Boolean satisfiability problem, with the distinction that instances are not necessarily in conjunctive normal form.

Hermann and Richoux [6] classified the complexity of $\text{MCSP}(\Gamma)$ with respect to polynomial-time many-one reductions. Since we are interested in a more fine-grained analysis of the complexity of $\text{MCSP}(\Gamma)$ we introduce a restricted form of reduction which only increases the number of variables within instances by an additive constant.

Definition 2. Let Γ and Δ be two constraint languages. A constant variable-reduction (*CV-reduction*) from $\text{MCSP}(\Gamma)$ to $\text{MCSP}(\Delta)$ is a computable function f from the instances of $\text{MCSP}(\Gamma)$ to the instances of $\text{MCSP}(\Delta)$ such that for every instance (V, ϕ) of $\text{MCSP}(\Gamma)$:

- $f((V, \phi))$ can be computed in $O(\text{poly}(|(V, \phi)|))$ time,
- (V, ϕ) is satisfiable if and only if $f((V, \phi))$ is satisfiable,
- $f((V, \phi)) = (V', \phi')$ where $|V'| = |V| + O(1)$.

We write $\text{MCSP}(\Gamma) \leq_{\text{CV}} \text{MCSP}(\Delta)$ as a shorthand for this reduction. In other words a CV-reduction is a polynomial-time many one reduction which only increases the number of variables by a constant. The utility of these reductions stems from the fact that if $\text{MCSP}(\Gamma)$ is solvable in $O(c^n \cdot \text{poly}(|I|))$ time for some constant $c \geq 1$, and if $\text{MCSP}(\Delta) \leq_{\text{CV}} \text{MCSP}(\Gamma)$, then $\text{MCSP}(\Delta)$ is also solvable in $O(c^n \cdot \text{poly}(|I|))$ time.

2.4 Closure Operators on Functions and Relations

Let R be a k -ary relation over a finite domain D . A unary function e over D is said to be an *endomorphism* of R if $(e(a_1), \dots, e(a_k)) \in R$ for every $(a_1, \dots, a_k) \in R$. In this case we also say that e *preserves* R or that R is *invariant* under e . This notion is extended to constraint languages in the obvious way. Given a constraint language Γ we let $\text{End}_D(\Gamma)$ denote the set of all endomorphisms over D of Γ . Similarly, given a set of unary functions E over D we let $\text{Inv}_D(E)$ denote the set of all relations over D that are invariant under E . Since the domain is typically clear from the context we usually just write $\text{End}(\Gamma)$ and $\text{Inv}(E)$. Sets of the form $\text{End}(\Gamma)$ and $\text{Inv}(E)$ are known as *endomorphism monoids* and *weak Krasner algebras*, respectively. Despite these rather enigmatic names they are in fact quite easy to grasp: $\text{End}(\Gamma)$ is a set of unary functions containing the identity function id which is closed under functional composition; $\text{Inv}(E)$ is a set of relations closed under monotone formulas [2]. The latter means that whenever $\Gamma \subseteq \text{Inv}(E)$ then $\text{Inv}(E)$ also contains all relations of the form $R(x_1, \dots, x_n) \equiv \phi$, where ϕ is a monotone formula over Γ . If we let $\langle \Gamma \rangle = \text{Inv}(\text{End}(\Gamma))$ we obtain the following Galois connection between weak Krasner algebras and endomorphism monoids.

Theorem 3 ([2]). *Let Γ and Δ be two constraint languages. Then $\Gamma \subseteq \langle \Delta \rangle$ if and only if $\text{End}(\Delta) \subseteq \text{End}(\Gamma)$.*

Using this Galois connection, Hermann and Richoux [6] proved that the complexity of $\text{MCSP}(\Gamma)$, up to polynomial-time many one reductions, is determined by the endomorphisms of Γ .

Theorem 4 ([6]). *Let Γ and Δ be two finite constraint languages. If $\text{End}(\Gamma) \subseteq \text{End}(\Delta)$ then $\text{MCSP}(\Delta)$ is polynomial-time many-one reducible to $\text{MCSP}(\Gamma)$.*

With this result they obtained a dichotomy theorem for $\text{MCSP}(\Gamma)$ for constraint languages Γ over arbitrary finite domains, proving that $\text{MCSP}(\Gamma)$ is NP-complete if and only if $\text{End}(\Gamma)$ does not contain a constant endomorphism, i.e. an endomorphism e which for some $j \in D$ satisfies $e(i) = j$ for all $i \in D$.

Theorem 5 ([6]). *Let Γ be constraint language. Then $\text{MCSP}(\Gamma)$ is NP-complete if and only if $\text{End}(\Gamma)$ does not contain a constant endomorphism.*

2.5 Restricted Closure Operators on Functions and Relations

We are now interested in closure operators based on quantifier-free monotone formulas. To get a similar Galois connection as in Theorem 6 we need a slight modification to the $\text{End}(\cdot)$ operator. An n -ary *partial function* f over D is a map $f : X \rightarrow D$ where $X \subseteq D^n$. Let $\text{dom}(f) = X$. If f and g are two partial functions then g is a *subfunction* of f if $\text{dom}(g) \subseteq \text{dom}(f)$ and $f(x_1, \dots, x_n) = g(x_1, \dots, x_n)$ for all $(x_1, \dots, x_n) \in \text{dom}(g)$. A set F of partial functions is *strong*, if, whenever $f \in F$, then F also contains all subfunctions of f . Given a set of partial functions F we let $\text{Strong}(F)$ denote the smallest strong set of partial functions containing F . A unary partial function e is said to be a *partial endomorphism* of a k -ary relation R if $(e(a_1), \dots, e(a_k)) \in R$ for all $(a_1, \dots, a_k) \in R$ such that $(a_1, \dots, a_k) \in \text{dom}(e)$. Again, this notion easily generalizes to constraint languages. Let $\text{pEnd}_D(\Gamma)$ denote the set of all partial endomorphisms over D to a constraint language Γ over D . As usual we omit the domain D when it is clear from the context. A set of the form $\text{pEnd}(\Gamma)$ is known as a *strong partial endomorphism monoid* [2] and is a strong, composition-closed set of unary partial functions containing the identity function. The utility of these definitions stems from the following: if E is a set of unary partial functions and if $\Gamma \subseteq \text{Inv}(E)$, then $\text{Inv}(E)$ also contains all relations $R(x_1, \dots, x_n) \equiv \phi$, where ϕ is a quantifier-free monotone formula over Γ [2]. For a constraint language Γ let $\langle \Gamma \rangle_{\neq} = \text{Inv}(\text{pEnd}(\Gamma))$. We have the following Galois connection.

Theorem 6 ([2]). *Let Γ and Δ be two constraint languages. Then $\Gamma \subseteq \langle \Delta \rangle_{\neq}$ if and only if $\text{pEnd}(\Delta) \subseteq \text{pEnd}(\Gamma)$.*

As will be made clear in Sect. 3, this Galois connection will allow us to obtain a corresponding result to Theorem 4, where we prove that the partial endomorphisms of a constraint language Γ determines the complexity of $\text{MCSP}(\Gamma)$ up to $O(c^{|V|})$ time complexity.

3 The Complexity of Monotone Constraint Satisfaction

By Theorem 5 we can for every $\Gamma \subseteq \text{Rel}_D$ easily determine whether $\text{MCSP}(\Gamma)$ is NP-complete or in P. We are interested in a more fine-grained analysis of the NP-complete MCSP problems with respect to CV-reductions. We first (in Sect. 3.1) prove that the partial endomorphisms of a constraint language determines the complexity of the MCSP problem with respect to CV-reductions. In Sect. 3.2 we then use partial endomorphism to obtain a full understanding of the applicability of CV-reductions for the MCSP problem.

3.1 Partial Endomorphisms and CV-reductions

We first prove an easy, but very important, theorem which gives a condition for obtaining a CV-reduction from one MCSP problem to another.

Theorem 7. *Let Γ and Δ be two finite constraint languages. If $\text{pEnd}(\Gamma) \subseteq \text{pEnd}(\Delta)$ then $\text{MCSP}(\Delta)$ is CV-reducible to $\text{MCSP}(\Gamma)$.*

Proof. Since $\text{pEnd}(\Gamma) \subseteq \text{pEnd}(\Delta)$ we can exploit the Galois connection to infer that $\Delta \subseteq \langle \Gamma \rangle_{\exists}$. This furthermore implies that every $R \in \Delta$ can be expressed as a quantifier-free monotone formula over Γ . Since both languages are finite we can easily find all such definitions in constant time, with respect to the size of Γ and Δ . Now let $I = (V, \phi)$ be an instance of $\text{MCSP}(\Delta)$. For each constraint $R_i(\mathbf{x}_i) \in \text{Constraints}(I)$ replace it by its equivalent quantifier-free monotone formula over Γ . Let $I' = (V, \phi')$ be the resulting instance. Then I' is satisfiable if and only if I is satisfiable. Since we do not introduce any fresh variables, and since the reduction runs in $O(\text{poly}(\|I\|))$ time, it follows that the reduction is a CV-reduction. \square

Since we are working over arbitrary finite domains we are interested in simplifying things whenever possible. The following theorem offers such a simplification whenever $\text{End}(\Gamma)$ contains a non-injective endomorphism, i.e., when the core-size of Γ is strictly smaller than $|D|$. The proof is simple and is therefore omitted.

Theorem 8. *Let Γ be a finite constraint language. Then (1) $\text{MCSP}(\Gamma) \leq_{\text{CV}} \text{MCSP}(\text{Core}(\Gamma))$ and (2) $\text{MCSP}(\text{Core}(\Gamma)) \leq_{\text{CV}} \text{MCSP}(\Gamma)$.*

3.2 Intervals of Strong Partial Endomorphism Monoids

By Theorem 7 we now have a relatively simple property for determining whether $\text{MCSP}(\Gamma)$ is CV-reducible to $\text{MCSP}(\Delta)$. This condition is sufficient to guarantee the existence of a CV-reduction, but as we will see in this section, there are many cases that are not covered. Assume e.g. that $\text{pEnd}(\Gamma) \subset \text{pEnd}(\Delta)$. Could it then still be the case that $\text{MCSP}(\Delta)$ is CV-reducible to $\text{MCSP}(\Gamma)$? In this section we obtain a complete understanding of when such CV-reductions are possible. Our main technical tool for accomplishing this is to study *intervals* of strong partial endomorphism monoids.

Definition 9. *Let $\text{End}(\Gamma)$ be an endomorphism monoid over D . The strong partial monoid interval of $\text{End}(\Gamma)$ is the set*

$$\mathcal{I}(\text{End}(\Gamma)) = \{\text{pEnd}(\Delta) \mid \text{End}(\Delta) = \text{End}(\Gamma)\}.$$

The smallest element in this set is given by $\text{Strong}(\text{End}(\Gamma))$ and the largest element by $\bigcup \mathcal{I}(\text{End}(\Gamma)) = \bigcup_{\text{pEnd}(\Delta) \in \mathcal{I}(\text{End}(\Gamma))} \text{pEnd}(\Delta)$. Hence, this set can indeed be viewed as a bounded interval. We illustrate this definition by an example.

Example 10. Consider the Boolean domain $D = \{1, 2\}$ and let $E = \{\text{id}\}$, i.e. the smallest Boolean endomorphism monoid consisting only of the unary projection function. Recall that $c_1 = \{(1)\}$, $c_2 = \{(2)\}$, let $c_{(1,2)} = \{(1, 2)\}$, and let e_1 and e_2 be the two partial functions $e_1(2) = 1$, $e_2(1) = 2$, which are undefined otherwise. Define $\text{pEnd}(\Gamma_1), \dots, \text{pEnd}(\Gamma_4)$ as:

- $\text{pEnd}(\Gamma_1) = \text{Strong}(\{\text{id}\})$, $\Gamma_1 = \{c_1, c_2\}$,
- $\text{pEnd}(\Gamma_2) = \text{pEnd}(\Gamma_1) \cup \{e_1\}$, $\Gamma_2 = \{c_1, c_{(1,2)}\}$
- $\text{pEnd}(\Gamma_3) = \text{pEnd}(\Gamma_1) \cup \{e_2\}$, $\Gamma_3 = \{c_2, c_{(1,2)}\}$, and
- $\text{pEnd}(\Gamma_4) = \text{pEnd}(\Gamma_1) \cup \text{pEnd}(\Gamma_2) \cup \text{pEnd}(\Gamma_3)$, $\Gamma_4 = \{c_{(1,2)}\}$.

Then one can prove that $\mathcal{I}(E) = \{\text{pEnd}(\Gamma_1), \text{pEnd}(\Gamma_2), \text{pEnd}(\Gamma_3), \text{pEnd}(\Gamma_4)\}$, and it is readily verified that the inclusions $\text{pEnd}(\Gamma_4) \supset \text{pEnd}(\Gamma_3) \supset \text{pEnd}(\Gamma_1)$ and $\text{pEnd}(\Gamma_4) \supset \text{pEnd}(\Gamma_2) \supset \text{pEnd}(\Gamma_1)$ hold.

In Example 10 one can also prove that $\text{MCSP}(\Gamma_1) \leq_{\text{CV}} \text{MCSP}(\Gamma_4)$. Due to the inclusion structure between these strong partial endomorphism monoids this furthermore implies that $\text{MCSP}(\Gamma_1), \dots, \text{MCSP}(\Gamma_4)$ are all CV-reducible to each other, and hence solvable within exactly the same $O(c^{|V|})$ running time. We are now interested in whether this holds when considering strong partial endomorphism monoid intervals over arbitrary finite domains. To accomplish this we first need a better characterization of the largest element $\bigcup \mathcal{I}(\text{End}(\Gamma))$.

Definition 11. Let $E = \text{End}(\Gamma)$ be an endomorphism monoid over $D = \{1, \dots, k\}$. The relation $E(D)$ is defined as $E(D) = \{(e(1), \dots, e(k)) \mid e \in E\}$.

The notation $E(D)$ is a mnemonic with the intended meaning that we are constructing a relation that is closed under every endomorphism in E .

Theorem 12. Let $E = \text{End}(\Gamma)$ be an endomorphism monoid over D . Then $\text{pEnd}(\{E(D)\}) = \bigcup \mathcal{I}(\text{End}(\Gamma))$.

Proof. First, we prove that $\text{pEnd}(\{E(D)\}) \in \mathcal{I}(E)$, i.e. that $\text{End}(\{E(D)\}) = E$. By definition $E(D)$ is closed under every function in E , so the inclusion $E \subseteq \text{End}(\{E(D)\})$ holds. Let $e \in \text{End}(\{E(D)\})$ and let $(e(1), \dots, e(k)) = (a_1, \dots, a_k)$. Observe that $(a_1, \dots, a_k) \in E(D)$ since e preserves $E(D)$. By definition of $E(D)$ there then exists $e' \in E$ such that $(e'(1), \dots, e'(k)) = (e(1), \dots, e(k)) = (a_1, \dots, a_k)$. Hence, $e \in E$.

Second, we prove that $\text{pEnd}(E(E)) \supseteq \text{pEnd}(\Delta)$ for any $\text{pEnd}(\Delta) \in \mathcal{I}(E)$. Let $e \in \text{pEnd}(\Delta)$. Assume towards contradiction that $e \notin \text{pEnd}(\{E(D)\})$. Then there exists $(b_1, \dots, b_k) \in E(D)$ such that $(e(b_1), \dots, e(b_k)) \notin E(D)$. Let $e' \in \text{End}(\{E(D)\}) = \text{End}(\Delta)$ be the total function satisfying $(e'(1), \dots, e'(k)) = (a_1, \dots, a_k)$. Observe that such a function must exist according to the definition of $E(D)$. Then define the unary function g as $g(i) = e(e'(i))$ for every $i \in D$. Clearly, g is a total function which does not preserve $E(D)$ since e is defined on $\text{img}(e')$, but this is a contradiction since $g \in \text{End}(\Delta) = \text{End}(\{E(D)\})$. \square

By combining Theorems 7 and 12 we obtain the following lemma.

Lemma 13. Let Γ be a finite constraint language over D and let $E = \text{End}(\Gamma)$. Then $\text{MCSP}(\{E(D)\}) \leq_{\text{CV}} \text{MCSP}(\Gamma)$.

We now have all the machinery in place to characterize the complexity of $\text{MCSP}(\Gamma)$. In particular, we want to prove the converse of Lemma 13, i.e. that

$\text{MCSP}(\Gamma)$ is CV-reducible to $\text{MCSP}(\{E(D)\})$. To prove this we first investigate the expressive power of the relation $E(D)$. Recall that Neq_D denotes the binary inequality relation over D .

Theorem 14. *Let Γ be a finite constraint language over D and let $D' \subseteq D$ be the domain of $\text{Core}(\Gamma)$. Then $\text{MCSP}(\{\text{Neq}_{D'}\}) \leq_{\text{CV}} \text{MCSP}(\Gamma)$.*

Proof. Let $k' = |D'|$, $k = |D|$, and let e be the homomorphism from Γ to $\text{Core}(\Gamma)$. Observe that $e \in \text{End}(\Gamma)$, $\text{img}(e) = D'$, and that there does not exist any $e' \in \text{End}(\Gamma)$ such that $|\text{img}(e')| < |\text{img}(e)|$. We prove that $\text{MCSP}(\{\text{Neq}_{D'}\})$ is CV-reducible to $\text{MCSP}(\{E(D)\})$, which according to Lemma 13 is sufficient to prove the claim. Let $t = (e(1), \dots, e(k)) \in E(D)$. Assume without loss of generality that $\{e(1), \dots, e(k')\} = D'$, i.e., that e is injective on the k' first arguments. Since $\text{img}(e) = D'$ this means that for every $i > k'$ there exists a $j \leq k'$ such that $t[i] = t[j]$. Let $h : \{k' + 1, \dots, k\} \rightarrow \{1, \dots, k'\}$ be a function satisfying $t[i] = t(h(i))$ for every $i \in \{k' + 1, \dots, k\}$. Let $R(x_1, \dots, x_{k'}) \equiv E(D)(x_1, \dots, x_{k'}, x_{h(k'+1)}, \dots, x_{h(k)})$, i.e. k' -ary relation obtained by identifying all arguments that are equal in the tuple t . We now claim that if $i \neq j$ then $t'[i] \neq t'[j]$ for every $t' \in R$. Assume to the contrary that there exists $t' \in R$ such that $t'[i] = t'[j]$ for some $i \neq j$. According to the definition of R this implies that there exists a tuple $t'' \in E(D)$ such that $t''[i] = t''[j]$, and, furthermore, that $t''[h(j')] = t''(j')$ for every $j' \in \{1, \dots, k'\}$. By letting $e'(1) = t''[1], \dots, e'(k) = t''[k]$, we see that $\text{img}(e') \subset \text{img}(e)$, a contradiction. Hence, all elements are distinct in every tuple $t' \in R$.

For the reduction, let $I = (V, \phi)$ be an instance of $\text{MCSP}(\{\text{Neq}_{D'}\})$. We introduce k' fresh variables $y_1, \dots, y_{k'}$ and introduce the constraint $R(y_1, \dots, y_{k'})$. For each constraint $\text{Neq}(x_i, x_j) \in \text{Constraints}(I)$ replace it by $(\text{Eq}(x_i, y_1) \wedge (\text{Eq}(x_j, y_2) \vee \dots \vee \text{Eq}(x_j, y_{k'}))) \vee \dots \vee (\text{Eq}(x_i, y_{k'}) \wedge (\text{Eq}(x_j, y_1) \vee \dots \vee \text{Eq}(x_j, y_{k'-1})))$. Let I' be the resulting instance over the variables $V \cup \{y_1, \dots, y_{k'}\}$. Clearly, if I is satisfiable then it is easy to find a satisfying assignment to I' . Similarly, if I' is satisfiable then one can apply e to the satisfying assignment to get an assignment over D' . It follows that the reduction is a CV-reduction since k' is a constant depending only on D and Γ . \square

This shows that $\text{MCSP}(\Gamma)$ is at least as hard as $\text{MCSP}(\{\text{Neq}_{D'}\})$ where D' is the domain of $\text{Core}(\Gamma)$. One might now wonder exactly how powerful the relation Neq_D is. Due to space constraints, we omit the proof, but it is in fact not difficult to see that whenever we have access to this appearingly simple relation, then $\text{MCSP}(\{\text{Neq}_D\})$ is as hard as $\text{MCSP}(\text{Rel}_D)$.

Theorem 15. *Let D be a finite domain. Then $\text{MCSP}(\text{Rel}_D) \leq_{\text{CV}} \text{MCSP}(\{\text{Neq}_D\})$.*

Put together, Theorems 14 and 15 imply that $\text{MCSP}(\Gamma)$ is always CV-reducible to $\text{MCSP}(\{\text{End}(\Gamma)(D)\})$, which results in the following corollary. The proof is straightforward and therefore omitted.

Corollary 16. *Let Γ and Δ be two finite constraint languages over D , with core-size c and d , respectively. If $d \leq c$ then $\text{MCSP}(\Delta) \leq_{\text{CV}} \text{MCSP}(\Gamma)$.*

4 Upper and Lower Bounds for the Complexity of Monotone Constraint Satisfaction

With Corollary 16 in Sect. 3 we now have a powerful condition for verifying whether $\text{MCSP}(\Gamma)$ is CV-reducible to $\text{MCSP}(\Delta)$. Moreover, since $\text{MCSP}(\Gamma)$ is solvable in $O(d^{|V|} \cdot \text{poly}(\|I\|))$, where d is the core-size of Γ , we have an obvious upper bound on the complexity of $\text{MCSP}(\Gamma)$ for all finite constraint languages. Proving lower bounds, i.e. the problem of proving that a problem is not solvable in $O(e^{\delta|V|})$ time for any $\delta < 1$, is much more challenging and usually requires stronger complexity theoretical assumptions than $\text{P} \neq \text{NP}$. Let SAT denote the Boolean satisfiability problem where instances are given as a tuple (V, ϕ) , where V is a set of variables and ϕ a conjunctive formula where each clause is a disjunction of positive and negative literals over V . The *strong exponential-time hypothesis* (SETH) is the conjecture that SAT is not solvable in $O(2^{\delta|V|})$ time for any $\delta < 1$ [4, 7]. Using the SETH we can not only prove that $\text{MCSP}(\text{Rel}_D)$ is not solvable in $O(2^{\delta|V|})$ time for any $\delta < 1$, unless the SETH fails, but that $\text{MCSP}(\text{Rel}_D)$ is not solvable in $O(|D|^{\delta|V|})$ for any $\delta < 1$.

Theorem 17. *Let D be a finite domain. If the SETH holds then $\text{MCSP}(\text{Rel}_D)$ is not solvable in $O(|D|^{\delta|V|})$ time for any $\delta < 1$.*

Proof. Assume that $\text{MCSP}(\text{Rel}_D)$ is solvable in $O(|D|^{\delta|V|})$ time for some $\delta < 1$. Let $I = (V, \phi)$ be an instance of SAT over the variables $V = \{x_1, \dots, x_n\}$ and the formula ϕ . Let $K \geq 1$ and $L = \lceil \frac{K}{\log_2(|D|)} \rceil$. The exact value of K , which is a constant depending on δ and D , will be determined later. Assume without loss of generality that $n \equiv 0 \pmod{K}$. We will partition V into subsets of size K and show that every such subset can be represented by L variables over D . Hence, let $V_1, \dots, V_{\frac{n}{K}}$ be such a partition of V . Let $f : \{1, \dots, n\} \rightarrow \{1, \dots, \frac{n}{K}\}$ be a function satisfying $f(i) = j$ if and only if $x_i \in V_j$. For every V_i introduce L fresh variables y_{i_1}, \dots, y_{i_L} , and observe that we in total require $\frac{n \cdot L}{K}$ new variables. Let $h : \{x_1, \dots, x_n\} \rightarrow \{1, \dots, K\}$ be a function which is injective on every V_i , i.e., every variable in a subset V_i of V is assigned a unique index from 1 to K .

Let b_D be an injective function from $\{0, 1\}^K$ to D^L . Such a function exists since by definition $2^K \leq |D|^L$. The purpose of b_D is to convert a K -ary Boolean sequence to an L -ary tuple over D . For each $i \in \{1, \dots, L\}$ define the L -ary relation $R_i^+ = \{b_D(x_1, \dots, x_K) \mid (x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_K) \in \{0, 1\}^K\}$, and the L -ary relation $R_i^- = \{b_D(x_1, \dots, x_K) \mid (x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_K) \in \{0, 1\}^K\}$.

Let $(\ell_{i_1} \vee \dots \vee \ell_{i_k}), \ell_{i_j} = x_{i_j}$ or $\ell_{i_j} = \neg x_{i_j}$, be a clause in ϕ . Let z_{i_1}, \dots, z_{i_k} such that $z_{i_j} = +$ if $\ell_{i_j} = x_{i_j}$ and $z_{i_j} = -$ if $\ell_{i_j} = \neg x_{i_j}$. For each literal ℓ_{i_j} let $V_{f(i_j)}$ be the partition corresponding to x_{i_j} and let $y_{f(i_j)_1}, \dots, y_{f(i_j)_L}$ be the corresponding variables over D . Then replace the clause $(\ell_{i_1} \vee \dots \vee \ell_{i_k})$ with

$$R_{h(x_{i_1})}^{z_{i_1}}(y_{f(i_1)_1}, \dots, y_{f(i_1)_L}) \vee \dots \vee R_{h(x_{i_j})}^{z_{i_j}}(y_{f(i_j)_1}, \dots, y_{f(i_j)_L}).$$

This reduction might appear to be complicated but essentially just follows the intuition that we can replace every variable set V_i with the corresponding

variables over D . Let I' be the resulting instance of $\text{MCSP}(\text{Rel}_D)$ over $\frac{n \cdot L}{K}$ variables. It is now easy to verify that I' is satisfiable if and only if I is satisfiable.

Hence, SAT can be solved in $O(|D|^{\delta \cdot \frac{n}{K} \cdot L})$ time. Now observe that $O(|D|^{\delta \cdot \frac{n}{K} \cdot L}) \subseteq O(|D|^{\delta \cdot \frac{n}{K} \cdot (\frac{K}{\log_2(|D|)} + 1)}) = O(|D|^{\delta \cdot (\frac{n}{\log_2(|D|)} + \frac{n}{K})}) = O(|D|^{\delta \cdot (n + \frac{n \cdot \log_2(|D|)}{K})}) = O(2^{\delta \cdot (n + \frac{n \cdot \log_2(|D|)}{K})}) = O(2^{n \cdot (\delta + \frac{\delta \cdot \log_2(|D|)}{K})})$. Since δ and $\log_2(|D|)$ are constants depending only on the domain D , SAT is solvable in $O(2^{n \cdot (\delta + \epsilon)})$ time for every $\epsilon > 0$, by choosing a large enough value of K . Hence, we can find an $\epsilon > 0$ such that SAT is solvable in $O(2^{n \delta'})$ for some $\delta' = \delta + \epsilon < 1$. This contradicts the SETH, and the result follows. \square

Let Γ be a constraint language over D and let D' be the domain of $\text{Core}(\Gamma)$. By Theorem 15 we know that $\text{MCSP}(\text{Rel}_{D'})$ is CV-reducible to $\text{MCSP}(\Gamma)$. Hence, we get the following corollary of Theorem 17.

Corollary 18. *Let Γ be a constraint language over a finite domain and let d be the core-size of Γ . If $\text{MCSP}(\Gamma)$ is NP-complete then $\text{MCSP}(\Gamma)$ is solvable in $O(d^{|V|})$ time but not in $O(d^{\delta |V|})$ for any $\delta < 1$, unless the SETH fails.*

We can also obtain lower bounds for $\text{CSP}(\text{Rel}_D)$ by using a similar proof strategy as in Theorem 17, but there is a caveat: relations in CSP instances are usually represented as list of tuples. Hence, if we replace a clause $(x_1 \vee \dots \vee x_n)$ by its corresponding constraint, then the resulting instance might be exponentially larger than the original SAT instance, since a clause of the form $(x_1 \vee \dots \vee x_n)$ can be compactly represented by a single tuple. However, we can obtain lower bounds for $\text{CSP}(\text{Rel}_D)$ vis-à-vis the complexity of $\text{CSP}(\text{Rel}_{\{1,2\}})$.

Corollary 19. *Assume that $\text{CSP}(\text{Rel}_{\{1,2\}})$ is not solvable in $O(2^{\delta |V|})$ time for any $\delta < 1$. Then, for any finite domain D , $\text{CSP}(\text{Rel}_D)$ is not solvable in $O(|D|^{\delta |V|})$ time for any $\delta < 1$.*

5 Concluding Remarks

In this paper we have obtained a complete classification of the worst-case time complexity of $\text{MCSP}(\Gamma)$. Most of the proofs make heavy use of the Galois connection between strong partial endomorphism monoids and weak Krasner algebras without existential quantification. Obtaining similar results for the CSP problem would likely be extremely difficult since one in this case needs to consider partial functions of arbitrary high arity [9, 11]. However, recent research in partial clone theory suggests that arity bounded sets of partial functions in many cases are expressive enough to characterize partial functions of arbitrary arity [12]. Hence, an interesting continuation of research would e.g. be to consider only binary or ternary partial functions, and investigate if such a restriction could be used increase our understanding of the worst-case time complexity of CSP and related problems.

Acknowledgments. The author is grateful towards Peter Jonsson for several helpful discussions regarding the content of this paper, and to the anonymous reviewers for many suggestions for improvement.

References

1. Bodirsky, M., Hermann, M., Richoux, F.: Complexity of existential positive first-order logic. In: Ambos-Spies, K., Löwe, B., Merkle, W. (eds.) CiE 2009. LNCS, vol. 5635, pp. 31–36. Springer, Heidelberg (2009)
2. Börner, F.: Basics of galois connections. In: Creignou, N., Kolaitis, P.G., Vollmer, H. (eds.) Complexity of Constraints. LNCS, vol. 5250, pp. 38–67. Springer, Heidelberg (2008)
3. Bulatov, A.: A dichotomy theorem for constraint satisfaction problems on a 3-element set. *J. ACM* **53**(1), 66–120 (2006)
4. Calabro, C., Impagliazzo, R., Paturi, R.: The complexity of satisfiability of small depth circuits. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 75–85. Springer, Heidelberg (2009)
5. Fargier, H., Marquis, P.: Disjunctive closures for knowledge compilation. *Artif. Intell.* **216**, 129–162 (2014)
6. Hermann, M., Richoux, F.: On the computational complexity of monotone constraint satisfaction problems. In: Das, S., Uehara, R. (eds.) WALCOM 2009. LNCS, vol. 5431, pp. 286–297. Springer, Heidelberg (2009)
7. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. *J. Comput. Syst. Sci.* **62**(2), 367–375 (2001)
8. Jeavons, P.: On the algebraic structure of combinatorial problems. *Theor. Comput. Sci.* **200**, 185–204 (1998)
9. Jonsson, P., Lagerkvist, V., Nordh, G., Zanuttini, B.: Complexity of SAT problems, clone theory and the exponential time hypothesis. In: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-2013), pp. 1264–1277 (2013)
10. Jonsson, P., Lagerkvist, V., Schmidt, J., Uppman, H.: Relating the time complexity of optimization problems in light of the exponential-time hypothesis. In: Csuhanj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part II. LNCS, vol. 8635, pp. 408–419. Springer, Heidelberg (2014)
11. Lagerkvist, V., Wahlström, M.: Polynomially closed co-clones. In: Proceedings of the 44th International Symposium on Multiple-Valued Logic (ISMVL-2014), pp. 85–90 (2014)
12. Lagerkvist, V., Wahlström, M., Zanuttini, B.: Bounded bases of strong partial clones. In: Proceedings of the 45th International Symposium on Multiple-Valued Logic (ISMVL-2015) (2015) (To appear)
13. Schaefer, T.: The complexity of satisfiability problems. In: Proceedings of the 10th Annual ACM Symposium on Theory Of Computing (STOC-78), pp. 216–226. ACM Press (1978)

Definability by Weakly Deterministic Regular Expressions with Counters is Decidable

Markus Latte^(✉) and Matthias Niewerth

Universität Bayreuth, Bayreuth, Germany
{markus.latte,matthias.niewerth}@uni-bayreuth.de

Abstract. We show that weakly deterministic regular expressions with counters (WDREs) —as they are used in XML Schema— are at most exponentially larger than equivalent DFAs. As a consequence, the problem, whether a given DFA is equivalent to any WDRE, is decidable in EXPSPACE.

1 Introduction

Deterministic or one-unambiguous regular expressions have been a topic of research since they were formally defined by Brüggemann-Klein and Wood in order to investigate a requirement in the ISO standard for the Standard Generalized Markup Language (SGML), where they were introduced to ensure efficient parsing. XML Schema, the current industry standard schema language for XML, also requires that the regular expressions defining its content models are deterministic. More precisely, an XML Schema is essentially a regular tree grammar in which right-hand sides of rules are *weakly deterministic regular expressions with counting* (WDREs) [6]. In the light that XML Schema is so wide-spread, it is surprising that WDREs are not well-understood. It is known that WDREs cannot define all regular word languages [6] but it is not yet known if it can be decided whether a given regular word language can be defined by a WDRE. In this paper, we prove that the latter problem is decidable in EXPSPACE.

Related Work. Brüggemann-Klein and Wood [3] first described an algorithm to decide whether there exist a deterministic regular expression — without counters but with Kleene stars — (DRE) for a given regular language. There has been further research on this BKW algorithm in [5, 14]. Gelade et al. analysed weakly and strongly deterministic regular expressions [6]. Bex et al. investigated algorithms for approximating regular languages by DREs [1]. Kilpeläinen and Tuhkanen provide PTIME algorithms for the membership problem of WDREs [10] and the problem checking whether a given regular expression with counters is weakly deterministic [11]. The latter complexity has been improved to linear time in [9]. Chen and Lu provide efficient algorithms checking whether an expression

M. Latte and M. Niewerth—Supported by grant number MA 4938/21 of the Deutsche Forschungsgemeinschaft (Emmy Noether Nachwuchsgruppe).

with counting is strongly deterministic [4]. There has been previous research on descriptonal complexity of DREs [12]. Dag Hovland investigated efficient matching algorithms for (deterministic) regular expressions with counters using automata with counters [7, 8].

Structure of the Proof. We limit the size of WDREs in several stages.

- Define some normal form that normalizes the structure of immediately nested counters. (Sect. 3)
- Partition all nodes in the parse tree of an expression in looping subexpressions and non-looping subexpressions. (Sect. 4)
- Show that a leaf-directed path in the parse tree consisting only of looping subexpression has at most polynomial length in the alphabet size when counting each block of immediately nested counters as length one. (Sect. 4)
- Show that each leaf-directed path in the parse tree has at most polynomially many non-looping subexpressions in the size of a minimal DFA. (Sect. 5)
- Define iterators to recover in a DFA the effective counters of a looping subexpression. (Sect. 6)
- Limit the number of immediately nested counters logarithmically and the upper bounds of each counter linearly in the size of a minimal DFA. (Sect. 7)

In Sect. 8, we connect all partial results to show that a minimal WDRE is at most exponentially larger than an equivalent DFA and in Sect. 9, we show a corresponding exponential lower bound for the size of WDREs.

2 Preliminaries

A *language* is a (possibly infinite) set of strings over a finite alphabet Σ . For any language L , we define $\text{first}(L) = \{a \in \Sigma \mid \exists w \in \Sigma^*. aw \in L\}$ and $\text{followlast}(L) = \{a \in \Sigma \mid \exists v \in L, w \in \Sigma^*. vaw \in L\}$. The *left quotient* of a language L by a word u is $u^{-1}L := \{w \mid uw \in L\}$ and by a language U is $U^{-1}L := \bigcup_{u \in U} u^{-1}L$. For all $N \subseteq \mathbb{N}$, let L^N abbreviate $\bigcup_{n \in N} L^n$ where L^n is the n -fold concatenation.

A (*deterministic, finite*) *automaton* (or *DFA*) \mathcal{A} is a tuple (Q, δ, q_0, F) , where Q is a set of states, $\delta : Q \times \Sigma \rightarrow Q$ is a partial transition function, q_0 is the initial state, and F is the set of final states. By δ^* we denote the extension of δ to strings (and languages), i.e., $\delta^*(q, w)$ is the state that can be reached from q by reading w . The *language* of \mathcal{A} is $L(\mathcal{A}) := \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$. We define the *size* of an automaton to be the number of its states.

We let \mathbb{N} and \mathbb{N}_+ denote the natural numbers with and without zero. Addition and multiplication are one-sidedly expanded on subsets of \mathbb{N} and understood pointwise. We set $I \otimes J := \{\sum_{k=1}^j i_k \mid j \in J \text{ and } i_k \in I \text{ for } 1 \leq k \leq j\}$ as the *product* of two subsets I, J of \mathbb{N} . Because $(2^{\mathbb{N}}, \otimes, \{1\})$ is a monoid, the product is canonically extended to lists, written as \otimes . The product \otimes is not commutative, as $[a \dots b] \otimes \{c\} = [ac \dots bc] \neq c[a \dots b] = \{c\} \otimes [a \dots b]$ if $a < b$ and $c > 1$.

We define $\mathbb{C}_{< \omega} := \{[c^- \dots c^+] \mid (c^-, c^+) \in \mathbb{N}^2 \setminus \{(0, 0)\} \text{ and } c^- \leq c^+\}$, $\mathbb{C}_\omega := \{\mathbb{N}, \mathbb{N}_+\}$, and $\mathbb{C} := \mathbb{C}_{< \omega} \cup \mathbb{C}_\omega$, as the sets of finite, infinite and all *counters*.

Each counter is just a subset of \mathbb{N} . Let $C \in \mathbb{C}$ be a counter. Then we use C^- and C^+ to denote $\min C$ and $\sup C$, respectively and C^\ominus to denote $\max(1, C^-)$. If $C^- = C^+$, we may denote the counter by the singleton set $\{C^-\}$. Lists of (finite) counters live in $[\mathbb{C}]$ and $[\mathbb{C}_{<\omega}]$, respectively. The concatenation symbol for counters is “.” or is omitted.

The *regular expressions* over Σ are defined as follows: ε , \emptyset and every Σ -symbol is a regular expression; and whenever r and s are regular expressions, then so are (rs) , $(r + s)$, and $(r)^C$, where C is a counter. For readability, we usually omit parentheses. Sometimes we write $r \cdot s$ instead of rs to emphasize that two expressions are concatenated. As syntactic sugar, $r^{\vec{C}}$ denotes the expression $(\dots((r)^{C_1})^{C_2} \dots)^{C_N}$ for any regular expression r and list $\vec{C} = C_1 \dots C_N \in [\mathbb{C}]$ of counters with $C_i \in \mathbb{C}$ for each i . The language of a regular expression r , denoted by $L(r)$, is defined as usual. For instance, $L(r^C) := L(r)^C$ for $C \in \mathbb{C}$. Thus, the usual Kleene star is a synonym for the counter \mathbb{N} in our setting. Although we do not explicitly consider the left-bounded interval $[c \dots]$ as a counter for $c \geq 2$, the expression $r^{[c \dots]}$ can be emulated by $(r^{\mathbb{N}^+})^{[c \dots c]}$. For each regular expression r , we let $\text{first}(r) = \text{first}(L(r))$ and $\text{followlast}(r) = \text{followlast}(L(r))$.

Intuitively, a regular expression is weakly deterministic if the following holds. When reading the input string from left to right, the expression always allows to match each symbol of that string uniquely against a position in the expression, without looking ahead. However, which counter is incremented might be ambiguous — in contrast to strong determinism [6].

Formally, let \bar{r} be the regular expression obtained from r by annotating every alphabet symbol with its position in the expression. For example, for $r = b^*a(b^*a)^*$ we have $\bar{r} = b_1^*a_2(b_3^*a_4)^*$. A regular expression r is *weakly deterministic* if for all $a \in \Sigma$ and all $wa_i v, wa_j v' \in L(\bar{r})$ the annotations i and j are equal. We denote the class of weakly deterministic regular expressions with counters by WDRE.

The expression $(a + b)^*a$ is not deterministic as already the first symbol in the string aaa could be matched by either the first or the second a in the expression. The equivalent expression $b^*a(b^*a)^*$, on the other hand, is deterministic. Brüggemann-Klein and Wood showed that not every (non-deterministic) regular expression is equivalent to a deterministic one [3]. Thus, semantically, not every regular language can be defined with a deterministic regular expression.

A WDRE r is *reentrant* iff whenever an alphabet symbol occurs in $\text{first}(r)$ and in $\text{followlast}(r)$ then both occurrences are justified by the same position in r . Intuitively, a reentrant WDRE is allowed to occur under a counter. It is easy to see that a WDRE r is reentrant iff r^* is a WDRE. We denote the set of all reentrant WDREs by WDRE° . The *size* of a WDRE r denoted by $|r|$ is the number of nodes of its parse tree plus the sum of the logarithms of the upper bounds of all its finite counters.

3 Normal Form

In this section, we will give a normal form for WDREs based on the following rewrite rules.

Lemma 1. *Let $a, b \in \mathbb{N}$ such that $a \leq b$, let $c, d \in \mathbb{N}_+$, $C_\omega \in \mathbb{C}_\omega$, and $C \in \mathbb{C}$.*

$$\begin{array}{ll}
 [1 \dots c] \otimes [0 \dots d] & = [0 \dots cd] & [0 \dots c] \otimes [a \dots b] & = [0 \dots bc] \\
 [1 \dots c] \otimes [1 \dots d] & = [1 \dots cd] & [0 \dots c] \otimes C_\omega & = \mathbb{N} \\
 [1 \dots c] \otimes C_\omega & = C_\omega & \mathbb{N} \otimes C & = \mathbb{N} \\
 \mathbb{N}_+ \otimes C & = [C^- \dots 2C^\ominus - 1] \otimes \mathbb{N}_+ & & (1)
 \end{array}$$

A challenge to our goal is that the operation \otimes does not preserve intervals in general as $[5 \dots 6] \otimes [3 \dots 4] = \{15, \dots, 18, 20, \dots, 24\}$.

Lemma 2. *Let $r \in \text{WDRE}$ and $\vec{C}, \vec{D} \in [\mathbb{C}]$. $L(r^{\vec{C}}) = L(r^{\vec{D}})$, if $\otimes \vec{C} = \otimes \vec{D}$.*

Definition 3. *Let r be a WDRE. The expression r is in normal form, iff the following conditions are true for every subexpression $s^{\vec{C}}$ with $\vec{C} = C_1 \dots C_N$ and every $i \leq N$:*

$$\begin{array}{ll}
 \varepsilon \in L(s) \rightarrow 0 \in C_1 & 0 \in C_i \rightarrow i = N \\
 1 \in C_i \ \& \ i < N \rightarrow 1 \notin C_{i+1} & C_i \in \mathbb{C}_\omega \rightarrow i = N
 \end{array}$$

Furthermore, \emptyset occurs in a WDRE r , iff $r = \emptyset$, ε occurs in a WDRE r , iff $r = \varepsilon$ and the counter $\{1\}$ does not occur.

As for the first condition, the empty words allows to lower C_1^- . The remaining conditions can be achieved by the rewriting rules from Lemma 1 via Lemma 2. Furthermore, it is well known that we need \emptyset only to represent the empty language. We can get rid of ε by replacing ε^C with $\varepsilon, \varepsilon \cdot s$ and $s \cdot \varepsilon$ with s , and $\varepsilon + s$ and $s + \varepsilon$ with $s^{[0 \dots 1]}$. We note that expressions in normal form can be slightly larger than minimal expressions, as the application of (1) can add at most one to the size for each counter above some \mathbb{N}_+ counter.

From now on, all considered WDREs are implicitly assumed to be in normal form. If we require some WDRE to be minimal, we mean a minimal WDRE among all WDREs in normal form.

4 Looping Subexpression

In this section, we define looping subexpressions and limit how many looping subexpressions can be nested into each other.

Definition 4. *The relation $_ \hookrightarrow _$ is inductively defined as a subset of $\text{WDRE}^\circ \times [\mathbb{C}] \times \text{WDRE}$.*

$$\begin{array}{ll}
 \text{Loop}_0 \frac{}{r \hookrightarrow^\varepsilon r} & \text{Loop}_+ \frac{s \hookrightarrow^{\vec{C}} r_i \quad i \in \{0, 1\}}{s \hookrightarrow^{\vec{C}} r_0 + r_1} \\
 \text{Loop}_{\text{ctr}} \frac{s \hookrightarrow^{\vec{C}} r}{s \hookrightarrow^{\vec{C} \cdot C} r^C} & \text{Loop}_\bullet \frac{s \hookrightarrow^{\vec{C}} r_i \quad \varepsilon \in L(r_{1-i}) \quad i \in \{0, 1\}}{s \hookrightarrow^{\vec{C}} r_0 r_1}
 \end{array}$$

Informally, $s \hookrightarrow^{\vec{C}} r$ states that s occurs under the counters \vec{C} in r and that it is possible to reenter s silently, i.e., without parsing the hypothetical input word for r any further. A subexpression s is looping (in r), denoted by $s \hookrightarrow r$, if $s \hookrightarrow^{\vec{C}} r$ for some $\vec{C} \in [\mathbb{C}]$.

We emphasize that $s \hookrightarrow^{\vec{C}} r$ does not imply that $s^{\vec{C}}$ is a subexpression of r , as the notation explicitly ignores some side branches in the parse tree. The negation of \hookrightarrow is denoted by $\not\hookrightarrow$.

Example 5. In $(a^{[2\dots 3]}b^{[0\dots 1]})^*$, the subexpressions a , $a^{[2\dots 3]}$, and $a^{[2\dots 3]}b^{[0\dots 1]}$ are looping while $b^{[0\dots 1]}$ is not. Moreover, b is looping in $b^{[0\dots 1]}$ although the counter exposes every reenter as pointless.

To restrict the maximal length of paths containing only looping subexpressions, we use the measure $\mu : 2^{\Sigma^*} \rightarrow 2^{\Sigma} \times 2^{\{\varepsilon\}} \times 2^{\Sigma}$ defined as follows.

$$L \mapsto (\text{first}(L), L \cap \{\varepsilon\}, \text{followlast}(L))$$

The implicit order is the lexicographic order over the inclusion where the left position is the most significant one. The measure is extended to regular expressions by considering their languages. For example, $a \cdot b^*$ is smaller than a^* although their followlast-sets are incomparable.

Lemma 6. Concerning the right argument of $-\hookrightarrow-$, the rule Loop_{ctr} decreases the measure weakly, while the rules Loop_+ and Loop_\bullet decrease the measure strictly. The rules are read upwards.

Proof. We use the notation as stated in the rules. With $\dot{\cup}$ we denote the union of two disjoint sets.

Rule Loop_{ctr} : We have $\text{first}(r^{\vec{C}}) = \text{first}(r)$, $\text{followlast}(r^{\vec{C}}) \supseteq \text{followlast}(r)$ and that $\varepsilon \in L(r)$ implies $\varepsilon \in L(r^{\vec{C}})$.

Rule Loop_+ : Because $r_0 + r_1 \in \text{WDRE}$, $\text{first}(r_0 + r_1) = \text{first}(r_0) \dot{\cup} \text{first}(r_1)$. Assume that $\text{first}(r_i) = \text{first}(r_0 + r_1)$. Then, $\text{first}(r_{1-i}) = \emptyset$, and thus $L(r_{1-i}) \subseteq \{\varepsilon\}$. However, the normal form excludes this situation.

Rule Loop_\bullet : The side condition “ $\varepsilon \in L(r_{1-i})$ ” entails that $\varepsilon \in L(r_0 r_1)$ iff $\varepsilon \in L(r_i)$. Moreover, the weak determinism of $r_0 r_1$ entails that

$$\begin{aligned} \text{first}(r_0 r_1) &= \text{first}(r_0) \underbrace{\dot{\cup} \text{first}(r_1)}_{\text{iff } \varepsilon \in L(r_0)}, \quad \text{and} \\ \text{followlast}(r_0 r_1) &= \underbrace{(\text{first}(r_1) \dot{\cup} \text{followlast}(r_0))}_{\text{iff } \varepsilon \in L(r_1)} \cup \text{followlast}(r_1), \end{aligned}$$

and is also responsible for the disjoint unions. So far, the measure is weakly decreasing. Because of the mentioned side condition and because r is in normal form, $L(r_{1-i})$ contains the empty word and a further word. Thus, $\text{first}(r_{1-i}) \neq \emptyset$. Therefore, if $i = 0$ then $\text{followlast}(r_0 r_1) \supsetneq \text{followlast}(r_0)$, and otherwise $\text{first}(r_0 r_1) \supsetneq \text{first}(r_1)$. \square

Theorem 7. *In any expression, the length of any path of therein looping subexpressions is bounded by $2|\Sigma|^2$ if every maximal group of immediately nested counters is counted as one.*

Proof. By Lemma 6 and the definition of μ . □

5 Non-Looping Subexpression

The following lemma will allow us to characterize languages of non-looping subexpressions by means of (simple) DFA operations that do not increase the size of an equivalent DFA.

Lemma 8. *If $s \not\prec r$ and u is a word that leads parsing in r to s , then*

$$u^{-1} R \cap (\text{first}(SZ)\Sigma^* \cup (SZ \cap \{\varepsilon\})) = S \cdot Z \tag{2}$$

where R stands for $L(r)$, S for $L(s)$ and $Z := (S^{-1}u^{-1}R) \setminus (\text{followlast}(S)\Sigma^*)$.

Proof. Let Y denote the language on the left side of (2).

Direction \subseteq . We silently use that r and s are weakly deterministic. Let $x \in Y$.

Since $ux \in R$, the parsing of $\text{first}(x)$ is handled by s or $x = \varepsilon \in S$. Thus, $x = yz$ for some $y \in S$ and some $z \notin \text{followlast}(S)\Sigma^*$. Therefore, $x = y \cdot y^{-1} u^{-1} (ux) \in S \cdot ((S^{-1}u^{-1}R) \setminus (\text{followlast}(S)\Sigma^*)) = S \cdot Z$.

Direction \supseteq . To show $u^{-1}R \supseteq SZ$, let $s_0, s_1 \in S$ and $z \in Z$ such that $us_1z \in R$.

The parsing of the factor s_1 in us_1z does not require the support of any rootward counter, because s is not looping in r . Therefore, s_1 can be replaced by any word in S , for instance by s_0 . In other words, $s_0z \in u^{-1}R$. □

Theorem 9. *Let r be a minimal WDRE, and let \mathcal{A} be an equivalent DFA with n states. Every leaf-directed path p in r hosts at most $n^3(|\Sigma| + 1)^2$ non-looping subexpressions. For any non-looping subexpression s on p , there exists a DFA with at most $n + 1$ states.*

Proofsketch. The length of paths that only have disjunctions and concatenations and take the right branch of each concatenation is limited by $n(|\Sigma| + 1)$. The basic idea is, that these paths can contain only n concatenations, as the language of the right side of a concatenation can be constructed in the automaton by just choosing a different initial state. Disjunctions restrict the set of first symbols, i.e., after each concatenation, there can be at most $|\Sigma|$ consecutive disjunctions.

Finally, whenever a path leading to a non-looping subexpression has some counter or uses the left branch of a concatenation, then Lemma 8 entails that the corresponding DFA essentially has less transitions than the DFA for the subexpression at the beginning of the path. Indeed, the left quotient and the intersection in (2) can be read as local modifications of R 's DFA. To unravel S from the right-hand side of (2), we remove those transitions, that leave some state reached after reading some word from S and using a symbol not in $\text{followlast}(S)$. As Z is nonempty in this case, such transitions have to exist.

Each automaton construction does not change the set of states. The only exception is caused by the reduction of the first-set in the case of disjunctions and of Lemma 8: the initial state is duplicated but without its incoming transitions. □

6 Iterators

In this section we will introduce iterators to connect the size of finite automata equivalent to some WDRE r such that $s \hookrightarrow^{\vec{C}} r$ with the size of automata accepting a unary representation of $\otimes \vec{C}$. This allows us in the next section to analyse the counters in \vec{C} without looking at the concrete language accepted by s .

Definition 10. An iterator for $r \in \text{WDRE}^\mathcal{Q}$ is a pair (x_0, x_1) of words such that

$$(x_0 x_1)^{\lfloor k/2 \rfloor} x_0^{k \bmod 2} \in L(r)^\ell \quad \text{iff} \quad k = \ell \quad \text{or} \quad \varepsilon \in L(r) \quad \text{and} \quad k \leq \ell$$

for all $k, \ell \in \mathbb{N}$.

Intuitively, reading both words of an iterator alternatively requires a hypothetical counter at the top of r , as the parsing cannot be continued within r .

Lemma 11. Let $r \in \text{WDRE}^\mathcal{Q}$ such that the topmost operator of r is not a counter. Then there is an iterator for r .

Proofsketch. If r is a letter, then we use (r, r) as iterator. If $r = r_0 + r_1$, we use (v_0, v_1) as iterator. And if $r = r_0 r_1$, an iterator is $(v_0 v_1, v_0 v_1)$. In the last two cases, v_i is a shortest word in $L(r_i) \setminus \{\varepsilon\}$. \square

The next technical lemma will be used to lift the iterator property from s to $r^{\vec{C}}$ whenever $s \hookrightarrow^{\vec{C}} r$. In the following theorem, we use the lemma to limit the size of DFAs accepting exactly the words of lengths from $\otimes \vec{C}$.

Lemma 12. If $s \hookrightarrow^{\vec{C}} r$ then $L(r) \cap L(s^*) \subseteq L(s^{\vec{C}})$.

Proof. The more general statement “ $L(r^{\vec{D}}) \cap L(s^*) \subseteq L(s^{\vec{C}\vec{D}})$ for all non-empty $\vec{D} \in [\mathbb{C}]$ such that $r^{\vec{D}}$ is in normal form” implies the claim with $[1 \dots 1]$ or $[0 \dots 1]$ as D depending on whether $\varepsilon \in L(r)$. For Loop_0 , the list \vec{C} is empty and $s = r$, yielding the statement. For the other rules, we prefer the notation of Definition 4.

Rule Loop_{ctr} : The induction hypothesis is instantiated with $C \cdot \vec{D}$ as \vec{D} .

Rule Loop_\bullet : Let $v \in L((r_0 r_1)^{\vec{D}}) \cap L(s^*)$. Because $v \in L(s^*)$ and because $(r_0 r_1)^{\vec{D}}$ is deterministic, the matching of v against $(r_0 r_1)^{\vec{D}}$ considers no leafs outside the parse tree of s , i.e., r_{1-i} is always matched against the empty word. Therefore, $L((r_0 r_1)^{\vec{D}}) \cap L(s^*) \subseteq L(r_i^{\vec{D}}) \cap L(s^*)$. The induction hypothesis yields the statement.

Rule Loop_+ : Let $v \in L((r_0 + r_1)^{\vec{D}}) \cap L(s^*)$. As with Loop_\bullet , the matching of v against $(r_0 + r_1)^{\vec{D}}$ considers no leafs outside the parse tree of s . Thus, the side branch r_{1-i} contributes empty words at the most. Because \vec{D} is not empty and since $(r_0 + r_1)^{\vec{D}}$ is in normal form, the existence of ε -contributions entails that $\otimes \vec{D}$ is downward closed. Hence, the omitted ε -contributions can be simulated by a smaller instance in $\otimes \vec{D}$. Thus, $v \in L(r_i^{\vec{D}}) \cap L(s^*)$. The induction hypothesis yields $v \in L(s^{\vec{C}\vec{D}})$. \square

Let $\mathbb{1}$ be some fixed letter. For each $n \in \mathbb{N}$, $\langle n \rangle$ stands for $\mathbb{1}^n$. The operation is extended to sets. A DFA *expresses* a subset N of \mathbb{N} iff its language is $\langle N \rangle$.

Theorem 13. *Let s, \vec{C} and r be such that $s \prec_{\vec{C}} r$ and s does not have a counter as topmost operation. If there is a DFA with n states for the language $L(r)$, then there is a DFA with $2n + 1$ states for $\langle \otimes \vec{C} \rangle$.*

Proof. The statement is trivial for $\vec{C} = \varepsilon$, therefore we assume $\vec{C} \neq \varepsilon$. Due to Lemma 11, the expression s has an iterator (x_0, x_1) . Let $g: \{\mathbb{1}\}^* \rightarrow \Sigma^*$ be the function $\mathbb{1}^k \mapsto (x_0 x_1)^{\lfloor k/2 \rfloor} x_0^{k \bmod 2}$. If $\varepsilon \in L(s)$, then the normal form entails that each counter in \vec{C} starts with 0 and thus $N := \otimes \vec{C}$ is downward closed. Independently of whether $\varepsilon \in L(s)$, Definition 10 therefore comes down to the statement: $g(\mathbb{1}^k) \in L(s)^N$ iff $k \in N$, for all $k \in \mathbb{N}$. A simple induction on $s \prec_{\vec{C}} r$ yields $L(s^{\vec{C}}) \subseteq L(r)$, because the additional side branches do not harm. Since $g(\mathbb{1}^k) \in L(s)^k \subseteq L(s^*)$ due to Definition 10, we obtain from Lemma 12 that $g(\mathbb{1}^k) \in L(r)$ implies $g(\mathbb{1}^k) \in L(s^{\vec{C}})$ for $k \in \mathbb{N}$. All together, $g^{-1}(L(r)) = g^{-1}(L(s)^{\vec{C}}) = \langle \otimes \vec{C} \rangle = \langle N \rangle$, where g^{-1} denotes the pre-image under g . The language $g^{-1}(L(r))$ is expressible by a $2n$ -state DFA, which can be shown by some kind of product construction of the DFA for $L(r)$ with the two-state DFA keeping track whether the next string should be x_0 or x_1 . □

7 Upper Bounds for Counters

In this section we give an upper bound on the number and values of counters of a minimal WDRE r based on the size n of an equivalent minimal DFA. We show that the upper bound of each finite counter is bounded linearly in n and that the number of immediately nested counters is bounded logarithmically in n . The former is established by showing, that each “large” upper bound can be replaced by a smaller one without changing the language.

We define $\mathfrak{h}: [\mathbb{C}] \rightarrow \mathbb{N}$ as $\mathfrak{h}(C_1 \dots C_N) := \prod_{i \leq N} C_i^{\ominus}$. We show in a series of technical lemmas, that if $s^C \prec_{\vec{C}} r$ then $\mathfrak{h}(\vec{C})$ iterations of C can be absorbed by the counters in \vec{C} . This will allow us to bound C linearly in $\mathfrak{h}(\vec{C})$, while we show that $\mathfrak{h}(\vec{C})$ is itself bounded linearly in the minimal DFA equivalent to r .

Lemma 14. *Let $C \in \mathbb{C}_{<\omega}$, and let $c^+, h \in \mathbb{N}_+$ such that $c^+ \geq C^-(1 + h)$, then $C \subseteq [C^- \dots c^+] \otimes (1 + h\mathbb{N})$.*

Proof. Let $n \in \mathbb{N}$, then $c^+(1 + hn) \geq C^-(1 + h)(1 + hn) \geq C^-(1 + h(n + 1))$, and thus the $(1 + hn)$ th and the $(1 + h(n + 1))$ th incarnation of $[C^- \dots c^+]$ are overlapping. Because $c^+ > 0$, the maximal number representable by the j th incarnation of $[C^- \dots c^+]$ is strictly growing with j . Therefore, the set $[C^- \dots c^+] \otimes (1 + h\mathbb{N})$ covers each number from C^- onwards. □

Lemma 15. *Let $C \in \mathbb{C}$ and $h \in \mathbb{N}$. Then, $(1 + C^{\ominus}h\mathbb{N}) \otimes C \subseteq C \otimes (1 + h\mathbb{N})$.*

Proof. Let $c \in C$ and $n_1, \dots, n_c, h \in \mathbb{N}$.

$$\sum_{i \leq c} 1 + C^\ominus h n_i = c + C^\ominus h \sum_{i \leq c} n_i \in C \otimes \left\{ 1 + h \sum_{i \leq c} n_i \right\} \subseteq C \otimes (1 + h\mathbb{N})$$

where “ \in ” holds because $\{c, C^\ominus\} \subseteq C$, and “ \subseteq ” because of \otimes ’s monotonicity. \square

The subsequent statements until Lemma 18 assume that each expression determines its position within its hosting expression. In this context, $r[s \leftarrow s']$ denotes the substitution of (the position of) s with s' in an expression r .

Lemma 16. *Let $r, s, s' \in \text{WDRE}$ and $\vec{C}, \vec{D} \in [\mathbb{C}]$ such that $s \hookrightarrow^{\vec{C}} r$. Then $L(s) \subseteq L(s')^{1+\mathfrak{h}(\vec{C}\vec{D})\mathbb{N}}$ implies $L(r) \subseteq L(r[s \leftarrow s'])^{1+\mathfrak{h}(\vec{D})\mathbb{N}}$.*

Proof. Induction on $s \hookrightarrow^{\vec{C}} r$ where \vec{D} is quantified internally. The list \vec{D} names hypothetical counter at the top of r . In the case of Loop_0 , the list \vec{C} is empty, and $s = r$. For the remaining rules, we prefer the notation of Definition 4.

Rule Loop_+ :

$$\begin{aligned} L(r) &\subseteq L(r_i + r_{1-i}) \\ &\subseteq L(r_i[s \leftarrow s'])^{1+\mathfrak{h}(\vec{D})\mathbb{N}} \cup L(r_{1-i}) \quad (\text{sem. of } + \text{ and IH}) \\ &\subseteq (L(r_i[s \leftarrow s']) \cup L(r_{1-i}))^{1+\mathfrak{h}(\vec{D})\mathbb{N}} \quad (\text{monotonicity of } _{}^{1+-}) \\ &\subseteq L(r[s \leftarrow s'])^{1+\mathfrak{h}(\vec{D})\mathbb{N}} \quad (\text{sem. of } + \text{ and } _{}^{- \leftarrow -}) \end{aligned}$$

Rule Loop_\bullet : The argument is analogous to Loop_+ ’s case but with additional use of $\varepsilon \in L(r_{1-i})$.

Rule Loop_{ctr} : As the rule addresses the counters $\vec{C}C$ instead of \vec{C} , the aimed implication is adjusted accordingly. The induction hypothesis for $C\vec{D}$ as \vec{D} entails that $L(r) \subseteq L(r[s \leftarrow s'])^{1+\mathfrak{h}(C\cdot\vec{D})\mathbb{N}}$. With help of Lemma 15 and the definition of \mathfrak{h} , we get $(1 + \mathfrak{h}(C \cdot \vec{D})\mathbb{N}) \otimes C \subseteq C \otimes (1 + \mathfrak{h}(\vec{D})\mathbb{N})$. Finally, the substitution $_{}^{-}[s \leftarrow s']$ commutes with $_{}^{-C}$. \square

Lemma 17. *Let $r, s, s' \in \text{WDRE}$, let $\vec{C} \in [\mathbb{C}]$, let and $C_\omega \in \mathbb{C}_\omega$ such that $s \hookrightarrow^{\vec{C}\cdot C_\omega} r$. Then $L(s) \subseteq L(s')^{1+\mathfrak{h}(\vec{C})\mathbb{N}}$ implies $L(r) \subseteq L(r[s \leftarrow s'])$.*

Proof. By induction on “ $s \hookrightarrow^{\vec{C}\cdot C_\omega} r$ ”. Eventually the rule Loop_{ctr} justifies the statement $s \hookrightarrow^{\vec{C}\cdot C_\omega} r_0^{C_\omega}$ with $s \hookrightarrow^{\vec{C}} r_0$ for some r_0 . For an empty list \vec{D} , Lemma 16 entails that $L(r_0) \subseteq L(r_0[s \leftarrow s'])^{1+\mathbb{N}}$. Because $(1 + \mathbb{N}) \otimes C_\omega \subseteq \mathbb{N}_+ \otimes C_\omega \subseteq C_\omega$ and because $_{}^{-C_\omega}$ commutes with the substitution, $L(r_0^{C_\omega}) \subseteq L(r_0^{C_\omega}[s \leftarrow s'])$. \square

Lemma 18. *Let $C_0 \in \mathbb{C}_{<\omega}$, $\vec{C} \in [\mathbb{C}]$, and $C_\omega \in \mathbb{C}_\omega$. If $s^{C_0} \hookrightarrow^{\vec{C}\cdot C_\omega} r$ for some minimal WDRE r , then $C_0^+ \leq C_0^\ominus \cdot (1 + \mathfrak{h}(\vec{C}))$.*

Proof. For the sake of contradiction, assume that $C_0^+ > c^+ := C_0^\ominus \cdot (1 + \mathfrak{h}(\vec{C}))$. Set $s^- := s^{[C_0^- \dots c^+]}$ and $r^- := r[s^{C_0} \leftarrow s^-]$. These expressions are weakly deterministic, because $[C_0^- \dots c^+] \in \mathbb{C}$. Due to monotonicity, $L(r^-) \subseteq L(r)$. For the other direction, Lemma 14 entails that $L(s^{C_0}) \subseteq L(s^-)^{1+\mathfrak{h}(\vec{C})\mathbb{N}}$. Thus, Lemma 17 yields $L(r) \subseteq L(r^-)$. Therefore, $L(r^-) = L(r)$ although $|r^-| < |r|$. \square

The previous restriction of counters is related with the transformation into the star normal form [2]. There, $(s_0s_1 + s_2)^*$ is rewritten as $(s_0 + s_1 + s_2)^*$ if $\varepsilon \in L(s_0s_1)$, for instance. One incarnation of s_0s_1 is replaced with two of $s_0 + s_1$ while the Kleene star can absorb arbitrarily many incarnations. Because \mathbb{C} also admits finite intervals, the absorption quantum here depends on the stack of counters under which the expression appears effectively.

Definition 19. A list $C_1 \dots C_N \in [\mathbb{C}]$ of counters propagates 0 iff $0 \in C_i$ together with $i \leq j$ implies $0 \in C_j$ for all $i, j \leq N$.

Lemma 20. Let $\vec{C} \cdot \vec{D} \in [\mathbb{C}]$ such that \vec{C} propagates 0. If $\otimes(\vec{C} \cdot \vec{D})$ is expressible by an n -state DFA, then $\mathfrak{h}(\vec{C}) < n$.

Proof. For every $C \in \mathbb{C}$ and $N \subseteq \mathbb{N}$, $\min((C \otimes N) \setminus \{0\}) = C^\ominus \cdot \max(1, \min(N))$. A simple induction using the 0-propagation entails that

$$\min\left(\otimes(\vec{C} \cdot \vec{D}) \setminus \{0\}\right) = \mathfrak{h}(\vec{C}) \cdot \min\left(\otimes \vec{D} \setminus \{0\}\right) \geq \mathfrak{h}(\vec{C}).$$

The number on the left is strictly bounded by $n - 1$ because the smallest non-empty word is reachable in the DFA without any loop. □

Lemma 21. Let $N \in \mathbb{N}$, let $C_1, \dots, C_N \in \mathbb{C}_{<\omega}$. If $\otimes_{i=1}^N C_i$ is expressible by an n -state DFA, then $\prod_{i=1}^N C_i^+ < n$.

Proof. Because $\otimes_{i=1}^N C_i$ is finite, the DFA lacks in loops. □

Theorem 22. Let $s, t \in \text{WDRE}$, $C \in \mathbb{C}_{<\omega}$, and $\vec{C} \in [\mathbb{C}]$ such that $t^C \hookrightarrow^{\vec{C}} s$. If s is minimal and $L(s)$ is expressible by an n -state DFA, then $C^+ \leq 4n$.

Proof. We remove all outermost counters of t . The normal form guarantees that these removed counters are finite. Thus, there is a list $\vec{C}_0 \in [\mathbb{C}_{<\omega}]$ such that $u \hookrightarrow^{\vec{C}_0} t$ where u denoted the obtained pruned subexpression. By transitivity, $u \hookrightarrow^{\vec{C}_0 \cdot C \cdot \vec{C}} s$. Because u does not have a counter as topmost operation, Theorem 13 entails that $\otimes(\vec{C}_0 \cdot C \cdot \vec{C})$ is expressible by a $(2n + 1)$ -state DFA.

If \vec{C} consists only of finite counters, then Lemma 21 bounds C^+ . Otherwise, \vec{C} can be written as $\vec{C}_{<\omega} \cdot C_\omega \cdot \vec{C}_{\omega+1}$ such that $\vec{C}_{<\omega} \in [\mathbb{C}_{<\omega}]$, $C_\omega \in \mathbb{C}_\omega$, and $\vec{C}_{\omega+1} \in [\mathbb{C}]$. The normal form entails that $C\vec{C}_{<\omega}$ propagates 0. By Lemmas 18 and 20, we have $C^+ \leq C^\ominus \cdot (1 + \mathfrak{h}(\vec{C}_{<\omega})) \leq 2\mathfrak{h}(C\vec{C}_{<\omega}) \leq 4n$. □

Theorem 23. Let $s, t, \vec{C}_0, \vec{C}_1$ be such that $t^{\vec{C}_0} \hookrightarrow^{\vec{C}_1} s$ and t does not have a counter as topmost operation. If $L(s)$ is expressible by an n -state DFA, then $|\vec{C}_0| \leq 2\lg(n) + 4$.

Proof. The normal form ensures that at most the last counter of \vec{C}_0 belongs to \mathbb{C}_ω . If so the last counter can be attributed to \vec{C}_1 . Formally, there are $\vec{C}_2 \in [\mathbb{C}_{<\omega}]$ and $\vec{C}_3 \in [\mathbb{C}]$ such that $\vec{C}_0\vec{C}_1 = \vec{C}_2\vec{C}_3$ and $|\vec{C}_2| \geq |\vec{C}_0| - 1$. The normal form also entails that \vec{C}_2 propagates 0. The set $\otimes \vec{C}_2 \otimes \otimes \vec{C}_3$ is expressible by

an $(2n + 1)$ -state DFA due to Theorem 13. Thanks to Lemma 20, $\mathfrak{h}(\vec{C}_2) \leq 2n$. Therefore, the number of counters which deny 0 or 1 is bounded by $\lg(2n)$. In front of, between, and after those counters, at most one other counter appears. Thus, $|\vec{C}_0| \leq |\vec{C}_2| + 1 \leq (2\lg(2n) + 1) + 1 \leq 2\lg(n) + 4$. \square

8 Upper Bound

Theorem 24. *Let \mathcal{A} be a DFA with n states such that $L(\mathcal{A})$ is expressible by a WDRE. Then a minimal WDRE for $L(\mathcal{A})$ is of size at most $2^{\mathcal{O}(|\Sigma|^4 n^3 \lg^2(n))}$ and all of its finite counters are bounded by $\mathcal{O}(n)$.*

Proof. We show that the parse tree of a minimal WDRE has depth at most $\mathcal{O}(|\Sigma|^4 n^3 \lg(n))$ and the upper bounds of (finite) counters are all bounded by $\mathcal{O}(n)$. As the parse tree is binary, this directly yields the claimed size bound.

Let r be a minimal WDRE in normal form equivalent to \mathcal{A} . By Theorem 9, any leaf-directed path in the parse tree of r can host at most $n^3(|\Sigma|+1)^2$ non-looping subexpressions. Furthermore, each non-looping subexpression has an automaton with at most $n+1$ states. By Theorem 23, each block of immediately nested counters has at most $2\lg(n+1) + 4$ counters. Combining this with Theorem 7 yields that any leaf-directed path consisting only of looping sub-expressions has length at most $2|\Sigma|^2(2\lg(n+1) + 4)$. Altogether, we get that the depth of the parse tree is bounded by $\mathcal{O}(|\Sigma|^4 n^3 \lg(n))$. Finally, we can apply Theorem 22 to bound the values of counters. Let C be a finite counter of r , such that $t^C \hookrightarrow^{\vec{C}} s$, where t is the subexpression below C and s is the lowest non-looping subexpression of r above C . Applying Theorem 22 yields that $C^+ \leq 4(n+1)$. \square

Corollary 25. *Let L be a regular language. If L is given by a DFA (a regular expression without counters, a regular expression with counters), it can be decided in EXPSPACE (2-EXPSPACE, 3-EXPSPACE), whether there is some WDRE for r .*

Proof. Compute a DFA \mathcal{A} for L with linearly (exponentially, double exponentially) many states. Enumerate [11] each WDRE up to the size bound of Theorem 24 and test whether its language is L : (i) Unravel all counters [6] while ignoring weak determinism. (ii) Test the obtained general regular expression against \mathcal{A} . \square

9 Lower Bound

We adapt an existing proof showing that WDREs without counters are exponentially larger than minimal DFAs from [13].

Theorem 26. *There exists a family of languages $(L_n)_{n \in \mathbb{N}}$ such that the minimal DFA for L_n has size $\Theta(n)$, and every minimal WDRE for L_n has size $2^{\Omega(n)}$.*

Proofsketch. As [13], we consider the finite languages $L_n = L((a + b)^{[0 \dots n]} \cdot b)$ for every $n \in \mathbb{N}$. The minimal DFA for L_n has $2n + 2$ states. By an inductive proof, it can be shown, that the minimal WDRE r_n for the language L_n is of the form $a \cdot r_{n-1} + b \cdot r_{n-1}^{[0 \dots 1]}$ which directly proves the assumption. \square

The main difference to the proof in [13] is that we have to consider counters in the inductive step. We note that Theorem 26 is independent of our normal form.

10 Conclusion

We have shown both an exponential upper and an exponential lower bound for the size of WDREs in terms of minimal DFA size. This easily gives an EXPSPACE upper bound for the decision problem, given a DFA does there exist an equivalent WDRE, solving an open problem from [6]. However, the complexity of this decision problem is still open, as we only have an NL lower bound that carries over from the problem for expressions without counters [14]. Especially, it is unclear, whether there is an adaption of the BKW algorithm presented in [3] that includes counters.

In [12, 13], the descriptonal complexity of DREs has been analysed. We believe, that the lower bounds for expression size can be transferred to WDREs, as the language families used in the proofs should not benefit from the use of counters.

References

1. Bex, G.J., Gelade, W., Martens, W., Neven, F.: Simplifying XML Schema: effortless handling of nondeterministic regular expressions. In: ACM SIGMOD, pp. 731–744. ACM (2009)
2. Brüggemann-Klein, A.: Regular expressions into finite automata. TCS **120**(2), 197–213 (1993)
3. Brüggemann-Klein, A., Wood, D.: One-unambiguous regular languages. Inf. Comput. **142**(2), 182–206 (1998)
4. Chen, H., Lu, P.: Checking determinism of regular expressions with counting. In: Yen, H.-C., Ibarra, O.H. (eds.) DLT 2012. LNCS, vol. 7410, pp. 332–343. Springer, Heidelberg (2012)
5. Czerwiński, W., David, C., Losemann, K., Martens, W.: Deciding definability by deterministic regular expressions. In: Pfenning, F. (ed.) FOSSACS 2013 (ETAPS 2013). LNCS, vol. 7794, pp. 289–304. Springer, Heidelberg (2013)
6. Gelade, W., Gyssens, M., Martens, W.: Regular expressions with counting: weak versus strong determinism. SIAM J. Comp. **41**(1), 160–190 (2012)
7. Hovland, D.: Regular expressions with numerical constraints and automata with counters. In: Leucker, M., Morgan, C. (eds.) ICTAC 2009. LNCS, vol. 5684, pp. 231–245. Springer, Heidelberg (2009)
8. Hovland, D.: The membership problem for regular expressions with unordered concatenation and numerical constraints. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 313–324. Springer, Heidelberg (2012)

9. Kilpeläinen, P.: Checking determinism of XML schema content models in optimal time. *Inf. Syst.* **36**(3), 596–617 (2011)
10. Kilpeläinen, P., Tuhkanen, R.: Towards efficient implementation of XML schema content models. In: *DocEng*, pp. 239–241. ACM (2004)
11. Kilpeläinen, P., Tuhkanen, R.: One-unambiguity of regular expressions with numeric occurrence indicators. *Inf. Comput.* **205**(6), 890–916 (2007)
12. Losemann, K., Martens, W., Niewerth, M.: Descriptive complexity of deterministic regular expressions. In: *Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464*, pp. 643–654. Springer, Heidelberg (2012)
13. Losemann, K., Martens, W., Niewerth, M.: Closure properties and descriptive complexity of deterministic regular expressions. Submitted, (2015)
14. Lu, P., Bremer, J., Chen, H.: Deciding determinism of regular languages. *TOCS* **57**(1), 1–43 (2014)

On the Complexity of Reconfiguration in Systems with Legacy Components

Jacopo Mauro^(✉) and Gianluigi Zavattaro

Department of Computer Science and Engineering,
University of Bologna / INRIA, Bologna, Italy
jmauro@cs.unibo.it

Abstract. In previous works we have proved that component reconfiguration in the presence of conflicts among components is non-primitive recursive, while it becomes poly-time if there are no conflicts and under the assumption that there are no components in the initial configuration. The case with non-empty initial configurations was left as an open problem, that we close in this paper by showing that, if there are legacy components that cannot be generated from scratch, the problem turns out to be PSpace-complete.

1 Introduction

Modern software systems are obtained as combination of software artefacts having complex interdependencies. Their composition, configuration and management is a difficult task, traditionally performed manually or by writing low level configuration scripts. Recently, many high level languages and tools like, for instance, TOSCA [17] or Engage [8] have been proposed to support the application manager in this difficult task. By adopting these tools, it is possible to describe the software components required to realise the system, define their interdependencies and specify the configuration actions to be executed to actually deploy an instance of the desired system. In some limited cases and under some specific assumptions (no circular component dependencies), such tools automatically synthesise the configuration actions to be executed. Automatic deployment is becoming more and more important for these tools especially due to the advent of virtualization technologies, like in Cloud Computing, that makes it possible to quickly acquire and release computing resources in order to deploy new software systems or reconfigure running applications on-demand.

In previous works [5,7,14] we have performed a rigorous and systematic analysis of the automatic deployment problem. We have proved that in general the problem is undecidable, it is non-primitive recursive if component interdependencies do not include numerical constraints, and it is poly-time if also conflicts among components are not considered. This last result was proved by restricting our attention on the deployment of an application from scratch, that is, by assuming that the initial configuration is empty. This result is of particular interest because it underpins the recent industrial trend of using the so

called “immutable servers” [16]. The application is divided in stateless components/services that are deployed on virtual machines. When a new version of the component is developed or the virtual machine needs updates (e.g., new security patches have to be installed), instead of upgrading in-place the virtual machine, a new one is created and the old one destroyed. According to this approach, since all the needed components can be freshly generated, the result proven in [14] shows that a new deployment can be efficiently computed simply generating a new configuration from scratch, without considering or reusing existing components.

Unfortunately the “immutable servers” approach has also some disadvantages. First of all, it requires that every application is carefully designed to ensure that important data is stored and not lost when the old servers are destroyed. System upgrades are usually slower because creating new virtual servers takes more time than performing an upgrade in-place. But, most importantly, this approach cannot be adopted in presence of legacy components, a scenario that often happens in practice due to software applications that for several reasons, like incompatibility with novel computing architectures or cost purposes, cannot be replaced and must be kept in-place.

Given these premises, the following question arises. How complex is the *reconfiguration* problem of deciding if a final configuration can be reached in the presence of components that cannot be switched off and re-deployed from scratch? The goal of this paper is to address this last question, proving that *reconfiguration* is no longer polynomial, but it turns out to be PSpace-complete.

More precisely, we first report the formalisation of the *reconfiguration* problem using the Aeolus component model adopted in [14] (Sect. 2). Then we show that the problem can be solved by performing a symbolic forward search of the new configurations that can be reached from a given initial one (Sect. 3). The symbolic approach allows for a finite representation of all the (possibly infinite) reachable configurations. Unfortunately, the number of possible symbolic configurations is exponential; we mitigate this blow up by adopting a nondeterministic polynomial-space visit of the (symbolic) search space. Finally, we show that it is not possible to significantly improve our algorithm as we prove that the reconfiguration problem is indeed PSpace-hard (Sect. 4). The proof is by reduction from the reachability problem in 1-safe Petri nets [2].

For space reasons, proofs are reported in [15].

2 Formalising the Reconfiguration Problem

In this section we recapitulate the fragment of the Aeolus model used to formally define the reconfiguration problem. This fragment of Aeolus [14] is exactly the one used by the planner Metis [13], a tool for finding deployment plans starting from an empty initial configuration integrated in an industrial deployment platform [6].¹ In the Aeolus model, a component is a grey-box showing relevant

¹ W.r.t. the Aeolus model [5], the fragment used by Metis does not allow the use of capacity constraints, conflicts, and multiple state changes.

internal states and the actions that can be acted on the component to change its state during (re)configuration. Each state activates provide-ports and require-ports representing functionalities that the component provides and needs. Active require-ports must be bound to active provide-ports of other components.

The problem that we address in this paper is verifying the existence of a plan (i.e., a correct sequence of configuration actions like component instantiation, binding, or internal state changes) that, given a universe of available components and an initial component configuration, leads to a configuration where a target component is in a given state.

As an example, consider the task of reconfiguring a system setting up a *MySQL master-slave replication* avoiding the downtime of an existing legacy MySQL database. Reconfigurations of this kind are frequent in practice, and are nowadays performed by system administrators who execute reconfiguration receipts that are part of their know-how. According to the Aeolus model, the problem can be formalised as follows. The involved components are two distinct database instances, one in master mode and one in slave mode. We assume to start from a configuration with only one legacy running instance, that will become the master in the new configuration. To activate the slave, a *dump* of the data stored in the master is needed. Moreover, the master has to authorise the slave. This is a circular dependency that is resolved by forcing a precise order in which the reconfiguration actions can be performed: the master first requires authentication of the slave that, subsequently, requires the dump from the master.

In Fig. 1, following the Aeolus model, we depict how to configure MySQL components as master or as slave. We assume the master component to be a legacy one, meaning that it can not be created from scratch but has to be used as deployed in the initial configuration. This is technically obtained setting a dummy state with no outgoing transitions as the initial one. In this way, no newly legacy component could be generated and moved in a state that is different from the dummy one. Apart from the initial dummy state, the master component

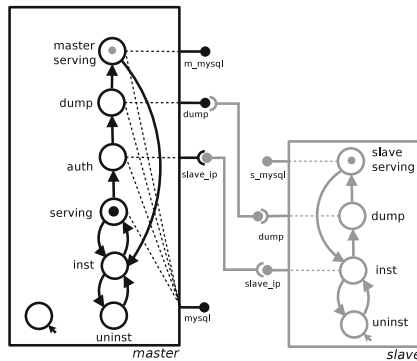


Fig. 1. MySQL master-slave instances (in black the initial configuration, in grey the parts added by the reconfiguration and the new state of the master) (Color figure online).

has 5 more states. The `uninst` state is followed by `inst` and `servng`. In `servng`, the master activates the provide-port `mysql` used by the clients to access the database service. When replication is needed, in order to enter the final `master servng` state, it first traverses the state `auth` that requires the IP address from the slave, and the state `dump` to provide the dump to the slave. The slave has instead 4 states, an initial `uninst` state and 3 states which complement those of the master during the replication process.

The formal definition of the Aeolus model is based on the notion of *component type*, used to specify the behaviour of a particular kind of component. In the following, \mathcal{I} denotes the set of port names and \mathcal{Z} the set of components.

Definition 1 (Component Type). *The set Γ of component types ranged over by $\mathcal{T}, \mathcal{T}_1, \mathcal{T}_2, \dots$ contains 4-tuples $\langle Q, q_0, T, D \rangle$ where:*

- Q is a finite set of states containing the initial state q_0 ;
- $T \subseteq Q \times Q$ is the set of transitions;
- D is a function from Q to a pair $\langle \mathbf{P}, \mathbf{R} \rangle$ of port names (i.e., $\mathbf{P}, \mathbf{R} \subseteq \mathcal{I}$) indicating the provide-ports and require-ports that each state activates. We assume that the initial state q_0 has no requirements (i.e., $D(q_0) = \langle \mathbf{P}, \emptyset \rangle$).

Configurations describe systems composed by components and their bindings. A binding connects a component providing a functionality with a component requiring it. Each component has a unique identifier, taken from the set \mathcal{Z} . A configuration, ranged over by $\mathcal{C}_1, \mathcal{C}_2, \dots$, is given by a set of available component types, a set of component instances in some state, and a set of bindings.

Definition 2 (Configuration). *A configuration \mathcal{C} is a quadruple $\langle U, Z, S, B \rangle$ where:*

- $U \subseteq \Gamma$ is the finite universe of the available component types;
- $Z \subseteq \mathcal{Z}$ is the set of the currently deployed components;
- S is the component state description, i.e., a function that associates to components in Z a pair $\langle \mathcal{T}, q \rangle$ where $\mathcal{T} \in U$ is a component type $\langle Q, q_0, T, D \rangle$, and $q \in Q$ is the current component state;
- $B \subseteq \mathcal{I} \times \mathcal{Z} \times \mathcal{Z}$ is the set of bindings, namely 3-tuples composed by a port, the component that provides that port, and the component that requires it; we assume that the two components are distinct.

Notation We write $\mathcal{C}[z]$ as a lookup operation that retrieves the pair $\langle \mathcal{T}, q \rangle = S(z)$, where $\mathcal{C} = \langle U, Z, S, B \rangle$. On such a pair we then use the postfix projection operators `.type` and `.state` to retrieve \mathcal{T} and q , respectively. Similarly, given a component type $\langle Q, q_0, T, D \rangle$, we use projections to decompose it: `.states`, `.init`, and `.trans` return the first three elements; `.P(q)` and `.R(q)` return the two elements of the $D(q)$ tuple. Moreover, we use `.prov` (resp. `.req`) to denote the union of all the provide-ports (resp. require-ports) of the states in Q . When there is no ambiguity we take the liberty to apply the component type projections to $\langle \mathcal{T}, q \rangle$ pairs. *Example:* $\mathcal{C}[z].\mathbf{R}(q)$ stands for the require-ports of component z in configuration \mathcal{C} when it is in state q .

As formalised below, a configuration is correct if all the active require-ports are bound to active provide-ports.

Definition 3 (Correctness). *Let us consider the configuration $\mathcal{C} = \langle U, Z, S, B \rangle$.*

We write $\mathcal{C} \models_{req} (z, r)$ to indicate that the require-port of component z , with port r , is bound to an active port providing r , i.e., there exists a component $z' \in Z \setminus \{z\}$ such that $\langle r, z', z \rangle \in B$, $\mathcal{C}[z'] = \langle T', q' \rangle$ and r is in $T'.\mathbf{P}(q')$.

The configuration \mathcal{C} is correct if for every component $z \in Z$ with $S(z) = \langle T, q \rangle$ we have that $\mathcal{C} \models_{req} (z, r)$ for every $r \in T.\mathbf{R}(q)$.

In Aeolus configurations evolve by means of (deployment) actions.

Definition 4 (Actions). *The set \mathcal{A} contains the following actions:*

- *stateChange(z, q, q') changes the state of the component $z \in \mathcal{Z}$ from q to q' ;*
- *bind(r, z_1, z_2) creates a binding between the provide-port $r \in \mathcal{I}$ of the component z_1 and the require-port r of z_2 ($z_1, z_2 \in \mathcal{Z}$);*
- *unbind(r, z_1, z_2) deletes the binding between the provide-port $r \in \mathcal{I}$ of the component z_1 and the require-port r of z_2 ($z_1, z_2 \in \mathcal{Z}$);*
- *new($z : T$) creates a new component of type T in its initial state. The new component is identified by a unique and fresh identifier $z \in \mathcal{Z}$;*
- *del(z) deletes the component $z \in \mathcal{Z}$.*

The execution of actions is formalised by means of a labelled transition system on configurations, which uses actions as labels.

Definition 5 (Reconfigurations). *Reconfigurations are denoted by transitions $\mathcal{C} \xrightarrow{\alpha} \mathcal{C}'$ meaning that the execution of $\alpha \in \mathcal{A}$ on the configuration \mathcal{C} produces a new configuration \mathcal{C}' . The transitions from a configuration $\mathcal{C} = \langle U, Z, S, B \rangle$ are defined as follows:*

$$\begin{array}{ll}
 \mathcal{C} \xrightarrow{\text{stateChange}(z, q, q')} \langle U, Z, S', B \rangle & \mathcal{C} \xrightarrow{\text{bind}(r, z_1, z_2)} \langle U, Z, S, B \cup \langle r, z_1, z_2 \rangle \rangle \\
 \text{if } \mathcal{C}[z].\text{state} = q \text{ and} & \text{if } \langle r, z_1, z_2 \rangle \notin B \\
 (q, q') \in \mathcal{C}[z].\text{trans} \text{ and} & \text{and } r \in \mathcal{C}[z_1].\text{prov} \cap \mathcal{C}[z_2].\text{req} \\
 S'(z') = \begin{cases} \langle \mathcal{C}[z].\text{type}, q' \rangle & \text{if } z' = z \\ \mathcal{C}[z'] & \text{otherwise} \end{cases} & \mathcal{C} \xrightarrow{\text{unbind}(r, z_1, z_2)} \langle U, Z, S, B \setminus \langle r, z_1, z_2 \rangle \rangle \\
 & \text{if } \langle r, z_1, z_2 \rangle \in B \\
 \\
 \mathcal{C} \xrightarrow{\text{new}(z:T)} \langle U, Z \cup \{z\}, S', B \rangle & \mathcal{C} \xrightarrow{\text{del}(z)} \langle U, Z \setminus \{z\}, S', B' \rangle \\
 \text{if } z \notin Z, T \in U \text{ and} & \text{if } S'(z') = \begin{cases} \perp & \text{if } z' = z \\ \mathcal{C}[z'] & \text{otherwise} \end{cases} \text{ and} \\
 S'(z') = \begin{cases} \langle T, T.\text{init} \rangle & \text{if } z' = z \\ \mathcal{C}[z'] & \text{otherwise} \end{cases} & B' = \{ \langle r, z_1, z_2 \rangle \in B \mid z \notin \{z_1, z_2\} \}
 \end{array}$$

A *deployment plan* is simply a sequence of actions that transform a correct configuration without violating correctness along the way.

Definition 6 (Deployment Plan). *A deployment plan \mathbf{P} from a correct configuration \mathcal{C}_0 is a sequence of actions $\alpha_1, \dots, \alpha_m$ s.t. there exists $\mathcal{C}_1, \dots, \mathcal{C}_m$ correct configurations s.t. $\mathcal{C}_{i-1} \xrightarrow{\alpha_i} \mathcal{C}_i$.*

In the following, exploiting the fact that reconfigurations are deterministic, we denote the deployment plan $\alpha_1, \dots, \alpha_m$ from C_0 also with the sequence of reconfigurations steps $C_0 \xrightarrow{\alpha_1} C_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_m} C_m$.

We now have all the ingredients to define the *reconfiguration problem*, that is our main concern: given a universe of component types and an initial configuration, we want to know whether and how it is possible to deploy at least one component of a given component type T in a given state q .

Definition 7 (Reconfiguration Problem). *The reconfiguration problem has as input a universe U of component types, an initial correct configuration C_0 , a component type T_t , and a target state q_t . The output is **yes** if there exists a deployment plan $P = C_0 \xrightarrow{\alpha_1} C_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_m} C_m$ s.t. $C_m[z] = \langle T_t, q_t \rangle$, for some component z in C_m . Otherwise, it returns **no**, stating that no such plan exists.*

As an example, considering Fig. 1, we can see that there are deployment plans that lead from the initial configuration (in black) to the final MySQL master-slave replication configuration. For instance, such a plan could start with the creation of the slave instance, followed by a state change to the `inst` state and the creation of a binding between the ports `slave_ip` of the two components. At this point, the master component can perform two state changes, reaching the `dump` state. Then, after another binding is established between the `dump` ports, the slave can be moved to its `servng` state by performing two state changes. Finally, the master can enter in the `master servng` state by performing a state change. Note that every action in the deployment plan will correspond to one or more concrete instructions. For instance, the state change from the `servng` to the `auth` state in the master corresponds to issue the command `grant replication slave on *.* to user@'slave_ip'`.

The addition of a dummy initial state to define the master component captures its legacy nature. Indeed, since no other state of the master component is reachable from the initial one, no component created from scratch can provide the same functionalities of the deployed master. For this reason, only the master component present in the initial configuration can be used to reach the target.

Notice that the restriction to consider one target state only in the definition of the reconfiguration problem is not limiting: one can require several target pairs $\langle T_t, q_t \rangle$ by adding dummy provide-ports enabled only by the components of type T_t in state q_t and a dummy target component that requires all such provides. For instance, Fig. 2 depicts the dummy target component that in `Inststate` requires both an active master and an active slave as needed in the MySQL master-slave reconfiguration discussed above.

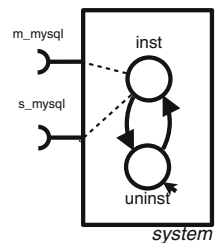


Fig. 2. Target

3 Solving the Reconfiguration Problem

In this section we present a nondeterministic polynomial space algorithm that resolves the reconfiguration problem, thus the problem is proved to be in PSpace

(as a consequence of the Savitch’s theorem [18] stating the equivalence between NPSpace and PSpace). The idea is to perform a nondeterministic forward exploration of the reachable configurations. This visit could be in principle arbitrarily long because infinitely many different configurations could be potentially reached. The main result that we prove in this section is that it is sufficient to consider a bounded amount of possibly reachable *abstract* configurations. In abstract configurations the bindings are not considered, but only the component type and state of the components are taken into account. Moreover, in abstract configurations, only the components present in the initial configuration are precisely represented, while for all the other components that are dynamically created, it is only considered the presence or absence of instances of components of type \mathcal{T} in state q , thus abstracting away from their precise number.

In order to abstract away from the bindings and consider only the component types and states, we define the following equivalence among configurations.

Definition 8 (Configuration Equivalence). *Two configurations $\langle U, Z, S, B \rangle$ and $\langle U, Z', S', B' \rangle$ are equivalent ($\langle U, Z, S, B \rangle \equiv \langle U, Z', S', B' \rangle$) iff there exists a bijective function ρ from Z to Z' s.t. $S(z) = S'(\rho(z))$ for every $z \in Z$.*

The research of the existence of the deployment plan is done on abstract configurations where bindings are not considered. We now show that this is not restrictive because every plan has a corresponding *normalised* plan where unbinding actions are absent and binding actions are generated as soon as possible.

Definition 9 (Normalised Deployment Plan). *A deployment plan $P = C_0 \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_m} C_m$ is normalised iff:*

- it does not contain unbind actions,
- if C_i for $i \in [1, m - 1]$ can be extended with a bind action then $\xrightarrow{\alpha_{i+1}}$ is a bind action,
- C_m cannot be extended with a bind action.

Lemma 1. *Given a deployment plan $P = C_0 \xrightarrow{\alpha_1} C_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_m} C_m$ there exists a normalised deployment plan $P' = C_0 \xrightarrow{\alpha'_1} C'_1 \xrightarrow{\alpha'_2} \dots \xrightarrow{\alpha'_n} C_n$ such that $C_n \equiv C_m$.*

In the remainder of the section, we assume a given universe U of component types; so we can consider that the set of distinct component type and state pairs $\langle \mathcal{T}, q \rangle$ is finite. Let k be its cardinality. Moreover, we assume a given initial configuration C_0 having the initial set of components Z_0 .

We are now ready to define our abstractions \mathcal{B} consisting of pairs of functions $\langle \mathcal{B}_i, \mathcal{B}_c \rangle$. Components are divided into two groups, those that were present in the *initial* configuration and those that were dynamically *created*: the first ones are precisely counted by the function \mathcal{B}_i , while for the second ones only the presence of a component type and state pair $\langle \mathcal{T}, q \rangle$ is checked by the function \mathcal{B}_c .

Definition 10 (Abstract Configuration). *An abstract configuration \mathcal{B} is a pair of functions $\langle \mathcal{B}_i, \mathcal{B}_c \rangle$ that associate to every pair $\langle \mathcal{T}, q \rangle$ respectively a natural number and a boolean value.*

It is immediate to see that (given a universe U of component types and an initial set Z_0 of components) the set of possible abstract configurations is finite: both functions have a domain bound by k , \mathcal{B}_c is a boolean function, and the sum of the values in the codomain of \mathcal{B}_i is bound by $|Z_0|$, i.e., the number of initial components, because such components can only be destroyed and not created.

A concretisation of an abstract configuration $\langle \mathcal{B}_i, \mathcal{B}_c \rangle$ is defined w.r.t. a set of initial components Z . These components occur according to the component type/state pairs counted by \mathcal{B}_i , while the other components satisfy the presence/absence indication of the boolean function \mathcal{B}_c . In the definition of concretisation we use the following notations: $\mathcal{C}_{\langle \mathcal{T}, q \rangle}^\#(Z)$ is the number of components in Z of type \mathcal{T} in state q in the configuration \mathcal{C} , while $Z - Z'$ is the set difference between two sets of components Z and Z' .

Definition 11 (Concretisation). *Given an abstract configuration $\mathcal{B} = \langle \mathcal{B}_i, \mathcal{B}_c \rangle$ and a set of components Z we say that a correct configuration $\mathcal{C} = \langle U, Z', S, B \rangle$ is one concretisation of \mathcal{B} w.r.t. Z if the following hold:*

- $\mathcal{B}_i(\langle \mathcal{T}, q \rangle) = \mathcal{C}_{\langle \mathcal{T}, q \rangle}^\#(Z)$;
- if $\neg \mathcal{B}_c(\langle \mathcal{T}, q \rangle)$ then $\mathcal{C}_{\langle \mathcal{T}, q \rangle}^\#(Z' - Z) = 0$;
- if $\mathcal{B}_c(\langle \mathcal{T}, q \rangle)$ then $\mathcal{C}_{\langle \mathcal{T}, q \rangle}^\#(Z' - Z) > 0$.

We denote with $\gamma(\mathcal{B}, Z)$ the set of concretisations of \mathcal{B} w.r.t. Z . We say that an abstract configuration \mathcal{B} is correct w.r.t. Z if it has at least one concretisation (formally $\gamma(\mathcal{B}, Z) \neq \emptyset$).

In the following, we usually consider concretisations w.r.t. the initial set of components Z_0 , and we simply use $\gamma(\mathcal{B})$ to denote $\gamma(\mathcal{B}, Z_0)$.

We now define the notion of deployment plan on abstract configurations and formalise its correspondence with *concrete* normalised plans.

Definition 12 (Abstract Deployment Plan). *We write $\mathcal{B} \rightarrow \mathcal{B}'$ with $\mathcal{B} \neq \mathcal{B}'$ if there exists $\mathcal{C} \xrightarrow{\alpha} \mathcal{C}'$ for some $\mathcal{C} \in \gamma(\mathcal{B})$ and $\mathcal{C}' \in \gamma(\mathcal{B}')$.*

A first lemma proves that each normalised deployment plan has a corresponding abstract version.

Lemma 2. *Given a normalised deployment plan $\mathbf{P} = \mathcal{C}_0 \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_m} \mathcal{C}_m$ there is an abstract deployment plan $\mathcal{B}_0 \rightarrow \dots \rightarrow \mathcal{B}_n$ s.t. $\mathcal{C}_0 \in \gamma(\mathcal{B}_0)$ and $\mathcal{C}_m \in \gamma(\mathcal{B}_n)$.*

The opposite correspondence (each abstract plan has at least one corresponding normalised *concrete* plan) is more complex to be formalised and proved. The intuition is that, given an abstract configuration $\mathcal{B} = \langle \mathcal{B}_i, \mathcal{B}_c \rangle$ that can be reached by an abstract plan, there exist normalised deployment plans able to reconfigure exactly the initial components as indicated by \mathcal{B}_i , and deploy an arbitrary number of instances of other components in the type and state indicated by the boolean function \mathcal{B}_c .

Lemma 3. *Given a correct configuration C_0 that cannot be extended with bind actions and an abstract deployment plan $\mathcal{B}_0 \rightarrow \dots \rightarrow \mathcal{B}_n = \langle \mathcal{B}_i, \mathcal{B}_c \rangle$ such that $C_0 \in \gamma(\mathcal{B}_0)$ then there is a normalised deployment plan $C_0 \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_{m-1}} C_m$ s.t.:*

- $C_m \in \gamma(\mathcal{B}_n)$;
- for all natural numbers $j_{\langle \mathcal{T}, q \rangle} > 0$, for every component type \mathcal{T} and state q such that $\mathcal{B}_c(\langle \mathcal{T}, q \rangle)$, then $C_m \overset{\#}{\langle \mathcal{T}, q \rangle} (Z_m - Z_0) = j_{\langle \mathcal{T}, q \rangle}$ where Z_m and Z_0 are the components of C_m and C_0 respectively.

Algorithm 1. Nondeterministic check for $C_0 = \langle U, Z_0, S, B \rangle$ and target \mathcal{T}_t, q_t

```

for all  $\langle \mathcal{T}, q \rangle$  pairs in the universe  $U$  do
     $\mathcal{B}_i(\langle \mathcal{T}, q \rangle) = C \overset{\#}{\langle \mathcal{T}, q \rangle} (Z_0)$ 
     $\mathcal{B}_c(\langle \mathcal{T}, q \rangle) = \text{False}$ 
    counter = 0
    while counter  $\leq |Z_0|^k * 2^k$  do                                 $\triangleright k$  is the number of  $\langle \mathcal{T}, q \rangle$  pairs in  $U$ 
        guess  $\mathcal{B}'_i, \mathcal{B}'_c$ 
        if  $\langle \mathcal{B}_i, \mathcal{B}_c \rangle \not\sim \langle \mathcal{B}'_i, \mathcal{B}'_c \rangle$  then return Failure
        if  $\mathcal{B}'_i(\mathcal{T}_t, q_t) > 0$  or  $\mathcal{B}'_c(\mathcal{T}_t, q_t)$  then return Success
        counter = counter + 1;  $\mathcal{B}_i = \mathcal{B}'_i$ ;  $\mathcal{B}_c = \mathcal{B}'_c$ 
    return Failure

```

In order to check if a solution to the reconfiguration problem exists, it is possible to consider all the possible abstract plans. This can be done using the nondeterministic Algorithm 1. Starting from the abstract representation $\langle \mathcal{B}_i, \mathcal{B}_c \rangle$ of the initial configuration C_0 , it performs a nondeterministic exploration of the reachable abstract configurations until either a configuration containing the target $\langle \mathcal{T}_t, q_t \rangle$ is reached or at least $K = |Z_0|^k * 2^k$ abstract steps have been considered, where $|Z_0|$ is the quantity of components of the initial configuration and k is the number of different $\langle \mathcal{T}, q \rangle$ pairs in the universe U . K is an upper bound to the number of different abstract configurations: $|Z_0|^k$ is an upper bound to the different combinations of states for the initially available components, while 2^k is the number of possible sets of $\langle \mathcal{T}, q \rangle$ pairs.

Assuming n the size of the input we have that $|Z_0| \leq n, k \leq n$ and therefore all the variables of the nondeterministic Algorithm 1 can be encoded in $O(n \log(n))$ space. For this reason (and for Savitch’s theorem [18]) we can conclude that the reconfiguration problem is in PSpace.

Theorem 1. *The reconfiguration problem is PSpace.*

4 The Reconfiguration Problem Is PSpace-hard

PSpace-hardness of the reconfiguration problem is proved by reduction from the reachability problem in 1-safe Petri nets, which is indeed known to be a PSpace-hard problem [2]. We start with some background on Petri nets.

A *Petri net* is a tuple $N = (P, T, \mathbf{m}_0)$, where P and T are finite sets of *places* and *transitions*, respectively. A finite multiset over the set P of places is called a *marking*, and \mathbf{m}_0 is the initial marking. Given a marking \mathbf{m} and a place p , we say that the place p contains a number of *tokens* equal to the number of instances of p in \mathbf{m} . A transition $t \in T$ is a pair of markings denoted with $\bullet t$ and t^\bullet . A transition t can fire in the marking \mathbf{m} if $\bullet t \subseteq \mathbf{m}$ (where \subseteq is multiset inclusion); upon transition firing the new marking of the net becomes $\mathbf{n} = (\mathbf{m} \setminus \bullet t) \uplus t^\bullet$ (where \setminus and \uplus are the difference and union operators for multisets, respectively). This is written as $\mathbf{m} \mapsto \mathbf{n}$. We use \mapsto^* to denote the reflexive and transitive closure of \mapsto . We say that \mathbf{m}' is *reachable from* \mathbf{m} if $\mathbf{m} \mapsto^* \mathbf{m}'$. A Petri net P is 1-safe if in every reachable marking every place has at most one token. Reachability of a specific marking \mathbf{m}_t from the initial marking \mathbf{m}_0 is PSPACE-complete for 1-safe nets [2].

We now consider a given 1-safe Petri net $N = \langle P, T, \mathbf{m}_0 \rangle$ and discuss how to encode it in Aeolus component types. We will use two types of legacy components: one modelling the places and one for the transitions. The simplest component type, denoted with \mathcal{T}_p and depicted in Fig. 3a, is the one used to model a place $p \in P$. Namely, a place p is encoded as one instance of \mathcal{T}_p . A token is present in p if the component of type \mathcal{T}_p is in the *on* state. There could be just one of these components deployed simultaneously. This can be obtained simply adding this component to the initial configuration in the *on* or *off* state, according to the initial marking, and make these two states non reachable from the initial state q_0 . The token could be created starting from the *off* state following a protocol consisting of providing the port a_p and then requiring the port on_p . Symmetrically, a token can be removed by providing the port b_p and then requiring the port off_p . The component provides the port on_p when it is in the *on* state, the port off_p when it is in the *off* state.

The transitions in T can be represented with a single component of type \mathcal{T}_T depicted in Fig. 3b. The uniqueness of this component is guaranteed, as done for \mathcal{T}_p , by adding it to the initial configuration and forbidding outgoing transitions

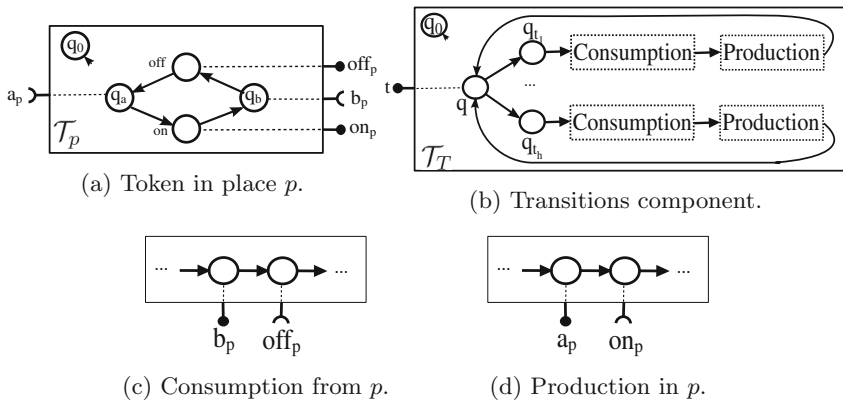


Fig. 3. 1-safe Petri net encoding

from the initial state q_0 . This component is assumed to be present in the initial configuration in state q . From this state it can nondeterministically select one transition t to fire, by entering a corresponding q_t state. The subsequent state changes can be divided into two phases: consumption and production. These phases respectively model the consumption of tokens from the places in the preset of t and the production of tokens in the places in the postset of t . The consumption and production of tokens have been already discussed above: consumption (see Fig. 3c) is obtained by providing and requiring the ports b_p and off_p , production (see Fig. 3d) by providing and requiring the ports a_p and on_p .

We now consider a marking \mathbf{m}_t of the 1-safe Petri net N . We can check whether \mathbf{m}_t is reachable in N by considering the following Aeolus reconfiguration problem. The initial configuration consists of an instance of \mathcal{T}_T , in state q , plus a component of type \mathcal{T}_p for every place p , in *on* or *off* depending on the initial marking \mathbf{m}_0 . The target to be considered consists of a configuration in which a port on_p is active for all places $p \in \mathbf{m}_t$, a port off_p is active for all places $p \notin \mathbf{m}_t$ and the port t is active indicating that no transition is currently in execution. Checking these requirements can be easily done, as explained in Sect. 2, by adding a dummy component having a target state requiring all the ports as explained above. Hence, we have the following.

Theorem 2. *The reconfiguration problem is PSpace-hard.*

5 Related Work and Conclusions

To the best of our knowledge, there is no work that formally studies the complexity of automatic reconfiguration of component systems. A significant part of the related literature focuses on the problem of dynamic re-allocation of resources, e.g., [3, 10]. Other works focus on the nature of the reconfiguration problem, like in [19] where a classification of the reconfiguration problems is made based on its causes, namely failures, system updates, and user requests. This work, however, does not consider the complexity of establishing the reconfiguration steps.

Different tools to compute the (optimal) final configuration exist, e.g., [4, 12]. However, all these approaches just focus on the target configuration to reach without computing the deployment steps. AI Planning Technologies [9] have been used to generate automatically the actions to reconfigure a system [1, 11]. However, these techniques have scalability issues. Conversely, tools like Metis [13, 14] or Engage [8] are able to compute the deployment steps needed to reach a target configuration but in simplified contexts: Metis imposes empty initial configurations while Engage forbids circular dependencies.

In this work we proved that extending these tools to deal also with reconfigurations may be too computationally expensive. Indeed, PSpace-completeness means that there are at least some cases where solving a reconfiguration problem requires a huge computational effort. For instance, our hardness proof shows that this can happen in the presence of legacy components that can not be recreated from scratch and may be required to perform cycles of deployment actions.

As a future work we plan to investigate limitations to be imposed to the Aeolus model (e.g., limiting the shape of the automata describing the components lifecycle) in order to have more efficient solutions for the reconfiguration problem. Another approach could be to relax completeness, by designing algorithms that could give negative answers even if a solution exists.

References

1. Chen, M., Poizat, P., Yan, Y.: Adaptive composition and QoS optimization of conversational services through graph planning encoding. In: *Web Services Foundations* (2014)
2. Cheng, A., Esparza, J., Palsberg, J.: Complexity results for 1-safe nets. *Theor. Comput. Sci.* **147**, 117–136 (1995)
3. Choi, H.W., Kwak, H., Sohn, A., Chung, K.: Autonomous learning for efficient resource utilization of dynamic VM migration. In: *ICS* (2008)
4. Cosmo, R.D., Lienhardt, M., Treinen, R., Zacchiroli, S., Zwolakowski, J., Eiche, A., Agahi, A.: Automated synthesis and deployment of cloud applications. In: *ASE* (2014)
5. Cosmo, R.D., Mauro, J., Zacchiroli, S., Zavattaro, G.: Aeolus: a component model for the cloud. *Inf. Comput.* **239**, 100–121 (2014)
6. Di Cosmo, R., Eiche, A., Mauro, J., Zavattaro, G., Zacchiroli, S., Zwolakowski, J.: Automatic Deployment of Software Components in the Cloud with the Aeolus Blender. Technical report, Inria Sophia Antipolis (2015)
7. Di Cosmo, R., Mauro, J., Zacchiroli, S., Zavattaro, G.: Component reconfiguration in the presence of conflicts. In: *ICALP* (2013)
8. Fischer, J., Majumdar, R., Esmaeilsabzali, S.: Engage: a deployment management system. In: *PLDI* (2012)
9. Ghallab, M., Nau, D.S., Traverso, P.: *Automated Planning - Theory and Practice*. Elsevier, Amsterdam (2004)
10. Gmach, D., Rolia, J., Cherkasova, L., Belrose, G., Turicchi, T., Kemper, A.: An integrated approach to resource pool management: Policies, efficiency and quality metrics. In: *DSN* (2008)
11. Herry, H., Anderson, P., Wickler, G.: Automated planning for configuration changes. In: *LISA* (2011)
12. Hewson, J.A., Anderson, P., Gordon, A.D.: A declarative approach to automated configuration. In: *LISA* (2012)
13. Lascu, T.A., Mauro, J., Zavattaro, G.: A planning tool supporting the deployment of cloud applications. In: *ICTAI* (2013)
14. Lascu, T.A., Mauro, J., Zavattaro, G.: Automatic component deployment in the presence of circular dependencies. In: *FACS* (2013)
15. Mauro, J., Zavattaro, G.: On the Complexity of Reconfiguration in Systems with Legacy Components. Technical report, INRIA Sophia Antipolis (2015)
16. Morris, K.: *Immutableserver* (2013). <http://martinfowler.com/bliki/ImmutableServer.html>
17. OASIS. Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0. <http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html>
18. Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.* **4**(2), 177–192 (1970)
19. Wang, S., Du, F., Li, X., Li, Y., Han, X.: Research on dynamic reconfiguration technology of cloud computing virtual services. In: *CCIS* (2011)

Eliminating Recursion from Monadic Datalog Programs on Trees

Filip Mazowiecki^(✉), Joanna Ochremiak, and Adam Witkowski

University of Warsaw, Warsaw, Poland
{f.mazowiecki, ochremiak, a.witkowski}@mimuw.edu.pl

Abstract. We study the problem of eliminating recursion from monadic datalog programs on trees with labels taken from an infinite alphabet. We show that the boundedness problem, i.e., determining whether a datalog program is equivalent to *some* nonrecursive one is undecidable but the decidability is regained if the descendant relation is disallowed. Under similar restrictions we obtain decidability of the problem of equivalence to a *given* nonrecursive program. We investigate the connection between these two problems in more detail.

1 Introduction

Among logics with fixpoint capabilities, one of the most prominent is datalog, which augments unions of conjunctive queries (positive existential first order formulae) with recursion. Datalog originated as a declarative programming language, but later found many applications in databases as a query language. The gain in expressive power does not, however, come for free. Compared to unions of conjunctive queries, evaluating a datalog program is harder [23] and basic properties such as containment or equivalence become undecidable [22].

Since the source of the difficulty in dealing with datalog programs is their recursive nature, the first line of attack in trying to optimize such programs is to eliminate the recursion. It is well-known that a nonrecursive datalog program can be rewritten as a union of conjunctive queries. The main focus of this paper is therefore the equivalence of recursive datalog programs to unions of conjunctive queries.

Example 1. The programs in this example work on databases that use binary predicates *likes* and *knows*, and a unary predicate *trendy*. First, consider the following pair of datalog programs:

$$\begin{array}{ll} \mathcal{P}_1 & \mathcal{P}'_1 \\ \text{buys}(X, Y) \leftarrow \text{likes}(X, Y) & \text{buys}(X, Y) \leftarrow \text{likes}(X, Y) \\ \text{buys}(X, Y) \leftarrow \text{trendy}(X), \text{buys}(Z, Y) & \text{buys}(X, Y) \leftarrow \text{trendy}(X), \text{likes}(Z, Y) \end{array}$$

The program \mathcal{P}_1 is recursive because its second rule refers to the predicate *buys*. It can be shown that \mathcal{P}_1 is equivalent to the nonrecursive program \mathcal{P}'_1 . Consider, on the other hand, the following pair of programs:

$$\begin{array}{ll}
\mathcal{P}_2 & \mathcal{P}'_2 \\
buys(X, Y) \leftarrow likes(X, Y) & buys(X, Y) \leftarrow likes(X, Y) \\
buys(X, Y) \leftarrow knows(X, Z), buys(Z, Y) & buys(X, Y) \leftarrow knows(X, Z), likes(Z, Y)
\end{array}$$

It can be shown that \mathcal{P}_2 is not equivalent to the nonrecursive program \mathcal{P}'_2 . Moreover, this program is not equivalent to any nonrecursive program.

The example above (taken from [19]) presents two approaches to eliminating recursion from datalog programs. Either we want to determine for a given datalog program if it is equivalent to *some* nonrecursive datalog program or decide whether a given datalog program is equivalent to a *given* nonrecursive program. These problems bear some similarities but in general they are distinct. The latter is decidable [12], while the former, called the *boundedness* problem, is not [16, 17].

Negative results for the full datalog fueled interest in its restrictions [4, 7, 8]. Important restrictions include *monadic* programs, using only unary predicates in the heads of rules; *linear* programs, with at most one use of an intensional predicate per rule; and *connected* programs, where within each rule all variables that are mentioned are connected to each other. Throughout this paper only monadic datalog programs are considered. In [13] Cosmadakis et al. show that for such programs the boundedness problem becomes decidable. Moreover, they use the same techniques to prove that the containment problem of two monadic datalog programs is decidable. These results suggest that under some additional assumptions the boundedness problem and the equivalence problem are more related.

In this paper we study connected, monadic datalog programs restricted to tree-structured databases. Our models are finite trees whose nodes carry labels from an infinite alphabet that can be tested for equality. Over such structures the problem of equivalence to a given union of conjunctive queries is known to be undecidable [1, 18]. We show that the boundedness problem is also undecidable. In some cases, however, we regain decidability of both problems in the absence of the descendant relation. On ranked trees we show that the equivalence and the boundedness problems become decidable (in 2-EXPTIME). On unranked trees we prove that the equivalence of a linear program to a non-recursive one is EXPSpace-complete. We finish with an analysis of the connection between the equivalence and the boundedness problems and show that under some assumptions they are equi-decidable.

Organization. In Sect. 2 we introduce datalog programs and some basic definitions. In Sect. 3 we deal with the problem of equivalence to a given nonrecursive datalog program. In Sect. 4 we analyze the boundedness problem. Finally, in Sect. 5 we explore the connection between the two approaches to eliminating recursion from datalog programs and show that under some assumptions the arising decision problems are equi-decidable. We conclude in Sect. 6 with possible directions for future research. Due to the page limit most of the proofs are moved to the appendix, available online.

2 Preliminaries

In this paper we work over finite trees labeled with letters from an infinite alphabet Σ . The trees are unranked by default, but we also work with ranked trees, in particular with words. We use the standard notation for axes: \downarrow, \downarrow_+ stand, respectively, for child and descendant relations. We assume that each node has one label. A binary relation \sim holds between nodes with identical labels and there is a unary predicate a for each $a \in \Sigma$, holding for the nodes labeled with a .

We begin with a brief description of the syntax and semantics of datalog; for more details see [2] or [9]. A **datalog program** \mathcal{P} over a relational signature S is a finite set of rules of the form $head \leftarrow body$ where $head$ is an atom over S and $body$ is a (possibly empty) conjunction of atoms over S written as a comma-separated list. All variables in the body that are not used in the head are implicitly quantified existentially. The **size of a rule** is the number of different variables that appear in it.

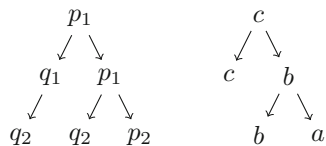
The relational symbols, or predicates, in S fall into two categories. **Extensional** predicates are the ones explicitly stored in the database; they are never used in the heads of rules. In our setting they come from $\{\downarrow, \downarrow_+, \sim\} \cup \Sigma$. The alphabet Σ is infinite, but the program \mathcal{P} uses only its finite subset which we denote by $\Sigma_{\mathcal{P}}$. **Intensional** predicates, used both in the heads and bodies, are defined by the rules.

The program is evaluated by generating all atoms (over intensional predicates) that can be inferred from the underlying structure (tree) by applying the rules repeatedly, to the point of saturation. Each inferred atom can be witnessed by a **proof tree**: an atom inferred by a rule r from intensional atoms A_1, A_2, \dots, A_n is witnessed by a proof tree with the root labeled by r , and n children which are the roots of the proof trees for atoms A_i (if r has no intensional predicates in its body then the root has no children).

There is a designated predicate called the **goal** of the program. We will often identify the goal predicate with the program, i.e., we write $\mathcal{P}(X)$ if the goal predicate of the program \mathcal{P} holds on the node X . When evaluated in a given database D , the program \mathcal{P} results in the unary relation $\mathcal{P}(D) = \{X \in D \mid \text{such that } \mathcal{P}(X) \text{ holds}\}$. If $\mathcal{P}(D) \subseteq \mathcal{Q}(D)$ for every database D then we say that the program \mathcal{P} is **contained** in the program \mathcal{Q} . If the containment holds both ways then the programs \mathcal{P} and \mathcal{Q} are **equivalent**.

Example 2. The program below computes the nodes from which one can reach some label a along a path where each node has a child with identical label and a descendant with label b (or has label b itself).

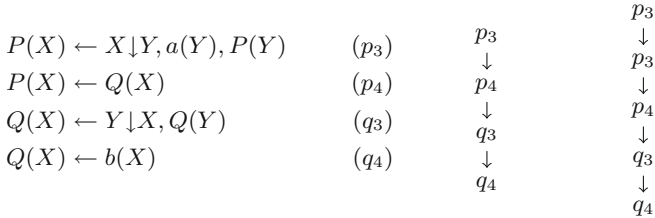
- $P(X) \leftarrow X \downarrow Y, P(Y), X \downarrow Y', X \sim Y', Q(X)$ (p₁)
- $P(X) \leftarrow a(X)$ (p₂)
- $Q(X) \leftarrow X \downarrow Y, Q(Y)$ (q₁)
- $Q(X) \leftarrow b(X)$ (q₂)



The intensional predicates are P and Q , and P is the goal. The proof tree shown in the center witnesses that P holds in the root of the tree on the right.

The notion of proof trees comes from papers on datalog over general structures (see e.g. [12]). As shown in Example 2 a proof tree illustrates how the program evaluates. If there is no restriction on the structures then for a given proof tree one can always find a model such that the proof tree witnesses a correct evaluation of the program. On tree structures this is no longer the case. Firstly, we allow only one label for every node, so rules like $P(X) \leftarrow a(X), b(X)$ cannot be satisfied. Moreover, nodes have a unique father. Because of this it is not easy to determine whether a given proof tree is a witness of an evaluation of the program on some model and it does not suffice to eliminate unsatisfiable rules. Proof trees for which such a model exists will be called **satisfiable proof trees**.

Example 3. The program below goes down a tree along a path labeled with a . Then it goes up the tree until it finds a node labeled with b .



The first proof tree is satisfiable, but the second proof tree is not satisfiable because it enforces both labels a and b on the same node.

In this paper we consider only **monadic** programs, i.e., programs whose intensional predicates are at most unary. Moreover, throughout the paper we assume that the programs do not use 0-ary intensional predicates. For general programs this is merely for the sake of simplicity: one can always turn a 0-ary predicate Q to a unary predicate $Q(X)$ by introducing a dummy variable X . For connected programs (described below) this restriction matters.

For a datalog rule r , let G_r be a graph whose vertices are the variables used in r and an edge is placed between X and Y if the body of r contains an atomic formula $X \downarrow Y$ or $X \downarrow_+ Y$. In G_r we distinguish a **head node** and **intensional nodes**. The latter are all variables from the body of r used by intensional predicates. A program \mathcal{P} is **connected** if for each rule $r \in \mathcal{P}$, G_r is connected¹.

Previous work on datalog on arbitrary structures often considered the case of connected programs [13, 16]. The practical reason is that real-life programs tend to be connected (cf. [3]). Also, rules which are not connected combine pieces of unrelated data, corresponding to the *cross product*, an unnatural operation in

¹ One could consider a definition allowing additionally nodes connected by the equality relation but we expect that this would be as hard as the disconnected case e.g. the main problem we leave open in Sect. 3, the equivalence of child-only non-linear programs, becomes undecidable by the results of [18] for boolean queries.

the database context. It seems even more natural to assume connectedness when working with tree-structured databases. We shall do so. We write $\text{Datalog}(\downarrow, \downarrow_+)$ for the class of connected monadic datalog programs, and $\text{Datalog}(\downarrow)$ for connected monadic programs that do not use the relation \downarrow_+ .

A datalog program is **linear** if the right-hand side of each rule contains at most one atom with an intensional predicate (proof trees for such programs are single branches). For linear programs we shall use the letter L , e.g., $L\text{-Datalog}(\downarrow)$ means linear programs from $\text{Datalog}(\downarrow)$. The program from Example 2 is connected, but not linear. The program from Example 3 is both connected and linear.

Conjunctive queries (CQs) are existential first order formulae of the form $\exists x_1 \dots x_k \phi$, where ϕ is a conjunction of atoms. We will consider **unions of conjunctive queries** (UCQs), corresponding to nonrecursive programs with a single intensional predicate (goal) which is never used in the bodies of rules. Since UCQs can be seen as datalog programs, we can speak of connected UCQs and as for datalog, we shall always assume connectedness. We denote the classes of connected queries by $\text{CQ}(\downarrow, \downarrow_+)$, $\text{CQ}(\downarrow)$, $\text{UCQ}(\downarrow, \downarrow_+)$, $\text{UCQ}(\downarrow)$, respectively.

3 Equivalence

The problem if two given programs \mathcal{P}, \mathcal{Q} are equivalent or one is contained in another one are called the **equivalence problem** and the **containment problem**. For datalog programs the containment problem can be reduced to the equivalence problem. Let \mathcal{P} be a datalog program and let \mathcal{Q} be a UCQ. Then $\mathcal{P} \subseteq \mathcal{Q}$ iff $\mathcal{P} \vee \mathcal{Q} \equiv \mathcal{Q}$. Notice that this reduction does not depend on the type of the programs (e.g., disallowing \downarrow_+ relation; or assuming linearity) but relies on the fact that datalog programs are closed under the disjunction.

The containment problem for datalog programs has been studied on trees in other contexts [1, 6, 15, 18]. In [18] containment of datalog programs in UCQs on data trees was analyzed in detail for boolean queries, which are queries that return the answer 'yes' if they are satisfied in some node of a given database, and the answer 'no', otherwise. More formally, a datalog program \mathcal{P} defines a *boolean query* $\mathcal{P}_{\text{Bool}}(D)$ which equals 1 iff $\mathcal{P}(D)$ is nonempty and 0 otherwise.

The containment problem is usually solved by considering the dual problem. For unary queries, it is the question whether there exist a database D and $X \in D$ such that $\mathcal{P}(X)$ and $\neg \mathcal{Q}(X)$, where $\neg \mathcal{Q} = D \setminus \mathcal{Q}(D)$. For boolean queries, it is the question if there exist a database D and $X, Y \in D$ such that $\mathcal{P}(X)$ and $\neg \mathcal{Q}(Y)$. For datalog programs over trees, if we allow the \downarrow_+ relation this distinction does not make much of a difference (intuitively because using \downarrow_+ one can move from a node X to any node Y). Thus a closer look at the proofs of Theorem 1 and Proposition 3 from [18] gives the following.

Proposition 1. *Over ranked and unranked trees the containment problem of $L\text{-Datalog}(\downarrow, \downarrow_+)$ programs in $\text{UCQ}(\downarrow, \downarrow_+)$ is undecidable.*

In the rest of this section we work only with fragments of datalog without the \downarrow_+ relation. We start with ranked trees.

Theorem 1. *The containment problem is 2-EXPTIME-complete for Datalog(\downarrow) over ranked trees. In the special case of words it is PSPACE-complete.*

The above result yields tight complexity bounds for the equivalence problem of Datalog(\downarrow) programs to UCQ(\downarrow) programs over ranked trees. To prove Theorem 1 we show that it suffices to solve this problem for trees over a finite alphabet. We then define automata that simulate the behavior of datalog programs, modifying the approach of [18]. The new construction gives better complexity results for non-linear programs².

In the rest of this section we focus on the equivalence problem of Datalog(\downarrow) programs to UCQ(\downarrow) programs over unranked trees. For the containment problem, this question was left open in [1].

For boolean queries, the containment problem of Datalog(\downarrow) programs in UCQ(\downarrow) programs was proved undecidable in [18]. Decidability was restored for the linear fragment, for which it was shown to be 2-EXPTIME-complete. We improve the complexity for unary queries using different techniques.

Theorem 2. *The containment problem of an L-Datalog(\downarrow) program in a UCQ(\downarrow) program is EXPSpace-complete over unranked trees.*

Unfortunately our approach does not generalize to the non-linear case. On the other hand, the proof of undecidability provided in [18] also cannot be adapted to work in our setting³. We leave the question of the decidability of containment for non-linear programs as an open problem.

We conclude with the following lemma (notice that we do not assume linearity).

Lemma 1. *The containment problem of UCQ(\downarrow) queries in Datalog(\downarrow) is in NPTIME over ranked and unranked trees.*

As a corollary of Theorem 2 and Lemma 1 we obtain the main result of this section. The lower bound is carried from the containment problem.

Theorem 3. *The equivalence problem of an L-Datalog(\downarrow) program to a UCQ(\downarrow) program is EXPSpace-complete over unranked trees.*

4 Boundedness

Consider a datalog program \mathcal{P} with a goal predicate P . By $\mathcal{P}^i(D)$ we denote the collection of facts about the predicate P that can be deduced from a database D by at most i applications of \mathcal{P} . More formally, $\mathcal{P}^i(D)$ is the subset of $\mathcal{P}(D)$

² In [18] the non-linear case required an additional exponential blow-up. However, the improvement of complexity is not caused by considering unary instead of boolean queries. It is easy to see that Theorem 1 holds also in the boolean case.

³ Indeed, the main idea of the undecidability proof is to use the UCQ \mathcal{Q} to find errors in the run of a Turing machine encoded by the program \mathcal{P} . If the nonrecursive query \mathcal{Q} is unary it can only find errors close to the node X , such that $\mathcal{P}(X)$.

derived using proof trees of height at most i , where the *height* of a tree is the length of the longest path from its root to a leaf. Then obviously

$$\mathcal{P}(D) = \bigcup_{i \geq 0} \mathcal{P}^i(D).$$

We say that the program \mathcal{P} is **bounded** if there exists a number n , depending only on \mathcal{P} , such that for any database D , we have $\mathcal{P}(D) = \mathcal{P}^n(D)$. Intuitively this means that the depth of recursion is independent of the input database⁴.

Each proof tree corresponds to a conjunctive query in a natural way. Therefore, we can always translate a datalog program to an equivalent, but possibly infinite, union of conjunctive queries. If the program is bounded then it is equivalent to a union of finitely many of these conjunctive queries. For full datalog it is known that the opposite implication is also true, i.e., a program is bounded iff it is equivalent to a (finite) UCQ [20]. The same holds for the class $\text{Datalog}(\downarrow)$.

Proposition 2. *Let $\mathcal{P} \in \text{Datalog}(\downarrow)$. Then \mathcal{P} is bounded iff it is equivalent to a union of conjunctive queries $\mathcal{Q} \in \text{UCQ}(\downarrow)$.*

We remark that the above characterization is based on the existence of so-called *canonical databases* for CQs (see e.g. [11]) in $\text{Datalog}(\downarrow)$. The following example shows that without canonical databases equivalence to some UCQ does not necessarily imply boundedness. It relies on the fact that \downarrow_+ is the transitive closure of \downarrow .

Example 4. The program $\mathcal{P} \in \text{Datalog}(\downarrow, \downarrow_+)$ on the left is not bounded – finding b in a tree can take arbitrarily long. The program \mathcal{P}' on the right is a UCQ equivalent to \mathcal{P} .

$$\begin{array}{l} \mathcal{P} \\ P(X) \leftarrow X \downarrow_+ Y, a(Y) \\ P(X) \leftarrow X \downarrow Y, Q(Y) \\ Q(X) \leftarrow X \downarrow Y, Q(Y) \\ Q(X) \leftarrow b(X) \end{array} \qquad \mathcal{P}' \qquad \begin{array}{l} P(X) \leftarrow X \downarrow_+ Y, a(Y) \\ P(X) \leftarrow X \downarrow_+ Y, b(Y) \end{array}$$

The problem if a given program \mathcal{P} is bounded is called the **boundedness problem**. We obtain a negative result for $\text{L-Datalog}(\downarrow, \downarrow_+)$.

Theorem 4. *The boundedness problem for $\text{L-Datalog}(\downarrow, \downarrow_+)$ is undecidable over words and ranked or unranked trees.*

In the following we work with fragments of datalog without the \downarrow_+ relation. For decidability results we use the automaton-theoretic approach of [13].

Theorem 5. *The boundedness problem for $\text{Datalog}(\downarrow)$ over words is in PSPACE.*

In the case of trees the same technique can be applied but the complexity increases.

⁴ Observe that we are only interested in the output on the goal predicate. This is why the property we consider is sometimes called the *predicate boundedness* [17].

Theorem 6. *The boundedness problem for Datalog(\downarrow) over ranked trees is in 2-EXPTIME.*

Over words, the relations \downarrow and \downarrow_+ are interpreted as the “next position” and the “following position”. Let X be a position in a word w . The n -neighbourhood of X in w is an infix of w , which begins on position $\max(1, X - n)$ and ends on position $\min(|w|, X + n)$. The following lemma is motivated by Proposition 3.2 of [13].

Lemma 2. *Let \mathcal{P} be a Datalog(\downarrow) program. Then \mathcal{P} is bounded iff there exists $n > 0$ such that for every word w and position X if $X \in \mathcal{P}(w)$ then $X \in \mathcal{P}(v)$, where v is the n -neighbourhood of X in w .*

Proof. (of Theorem 5) A word w such that for some position X in w we have $X \in \mathcal{P}(w)$ but $X \notin \mathcal{P}(v)$, where v is the n -neighbourhood of X in w will be called an n -witness. By Lemma 2 a Datalog(\downarrow) program \mathcal{P} is unbounded iff there exist n -witnesses for arbitrarily big $n > 0$.

Consider a Datalog(\downarrow) program \mathcal{P} . Let Σ_0 be an alphabet that contains the set of labels used explicitly in the rules of \mathcal{P} together with N “fresh” labels, where N is the size of the biggest rule in \mathcal{P} . It is known [18] (and easy to verify) that any word w can be relabeled so that the obtained word w' uses only labels from Σ_0 , and for each position X we have that $X \in \mathcal{P}(w)$ iff $X \in \mathcal{P}(w')$. This is also true with respect to infixes, i.e., for every infix v of w , and every position X it holds that $X \in \mathcal{P}(v)$ iff $X \in \mathcal{P}(v')$, where v' is the corresponding infix of w' . Hence, we can verify the existence of n -witnesses over the finite alphabet Σ_0 .

In the proof of Theorem 1 a nondeterministic automaton is introduced that recognizes words over the alphabet Σ_0 satisfying \mathcal{P} . More precisely, the constructed automaton $\mathcal{A}_{\mathcal{P}}$ works over the alphabet $\Sigma_0 \times \{0, 1\}$, and accepts a word w iff it has exactly one position X marked with 1 such that $X \in \mathcal{P}(w)$. We denote the language recognized by $\mathcal{A}_{\mathcal{P}}$ by $L(\mathcal{A}_{\mathcal{P}})$. The size of this automaton is exponential in the size of \mathcal{P} .

Similarly, we obtain an automaton $\mathcal{N}_{\mathcal{P}}$ recognizing these words over the alphabet $\Sigma_0 \times \{0, 1\}$ which have exactly one position marked with 1 but do not belong to $L(\mathcal{A}_{\mathcal{P}})$. The size of $\mathcal{N}_{\mathcal{P}}$ is also exponential in the size of \mathcal{P} (there is no exponential blow up because the constructions in the proof of Theorem 1 go through alternating automata) and the language it recognizes will be denoted $L(\mathcal{N}_{\mathcal{P}})$. Note that this language is closed under infixes containing the marked position.

We define a nondeterministic automaton $\mathcal{B}_{\mathcal{P}}$ which accepts exactly those words belonging to $L(\mathcal{A}_{\mathcal{P}})$ which have an infix that belongs to $L(\mathcal{N}_{\mathcal{P}})$. The states and transitions of $\mathcal{B}_{\mathcal{P}}$ are the states and transitions of the product automaton $\mathcal{A}_{\mathcal{P}} \times \mathcal{N}_{\mathcal{P}}$ together with the states and transitions of two copies of the automaton $\mathcal{A}_{\mathcal{P}}$ denoted $\mathcal{A}_{\mathcal{P}}^1$ and $\mathcal{A}_{\mathcal{P}}^2$. Let q_{init} be the initial state of $\mathcal{N}_{\mathcal{P}}$. For each state q of $\mathcal{A}_{\mathcal{P}}^1$ we add to $\mathcal{B}_{\mathcal{P}}$ an epsilon transition from the state q to the state (q, q_{init}) of the product automaton. Now, let F be the set of final states of $\mathcal{N}_{\mathcal{P}}$. For each state q of $\mathcal{A}_{\mathcal{P}}^2$ and each $q_{fin} \in F$ we add to $\mathcal{B}_{\mathcal{P}}$ an epsilon transition from the state (q, q_{fin}) to q . The initial state of $\mathcal{B}_{\mathcal{P}}$ is the initial state of $\mathcal{A}_{\mathcal{P}}^1$ and the final

states of $\mathcal{B}_{\mathcal{P}}$ are the final states of $\mathcal{A}_{\mathcal{P}}^2$. Hence, an accepting run of the automaton $\mathcal{B}_{\mathcal{P}}$ starts in $\mathcal{A}_{\mathcal{P}}^1$, moves to the product automaton at some point, reads an infix that belongs to $L(\mathcal{N}_{\mathcal{P}})$ and finally goes to $\mathcal{A}_{\mathcal{P}}^2$ to accept.

Let N be the number of states of the product automaton $\mathcal{A}_{\mathcal{P}} \times \mathcal{N}_{\mathcal{P}}$ plus 1. Suppose that $\mathcal{B}_{\mathcal{P}}$ accepts an N -witness w . Then, due to the pumping lemma, it accepts n -witnesses for arbitrarily big $n > 0$. To end the proof show that checking whether $\mathcal{B}_{\mathcal{P}}$ accepts some N -witness is in NLOGSPACE in the size of the automata $\mathcal{A}_{\mathcal{P}}$ and $\mathcal{N}_{\mathcal{P}}$ (i.e., in PSPACE in the size of \mathcal{P}).

An N -witness is a word that belongs to $L(\mathcal{A}_{\mathcal{P}})$ but the N -neighbourhood of the position marked with 1 belongs to $L(\mathcal{N}_{\mathcal{P}})$. The NLOGSPACE algorithm simulates a run of the automaton $\mathcal{B}_{\mathcal{P}}$. The size of $\mathcal{B}_{\mathcal{P}}$ is exponential in the size of \mathcal{P} but its states and transitions can be generated on the fly in polynomial space. The algorithm guesses a state from the $\mathcal{A}_{\mathcal{P}} \times \mathcal{N}_{\mathcal{P}}$ part and checks if it is reachable from the initial state. This is a simple reachability test which is in NLOGSPACE. Then it guesses some run of the $\mathcal{A}_{\mathcal{P}} \times \mathcal{N}_{\mathcal{P}}$ part, counts the number of transitions done before the one marked with 1, and ensures that it is at least N . After the transition marked with 1 it ensures that the automaton makes at least N more transitions before leaving the $\mathcal{A}_{\mathcal{P}} \times \mathcal{N}_{\mathcal{P}}$ part. For both of these counting procedures we need $\log(N)$ tape cells. Finally, the algorithm performs a second reachability test to check if the automaton can reach a final state.

There are three possible ways of how an N -witness v may look like. For simplicity, the algorithm described above does not deal with the case when the N -neighbourhood that belongs to $L(\mathcal{N}_{\mathcal{P}})$ is shorter than $2N + 1$ (which can happen if it begins at the first position of w or ends at the last position of w). Those possibilities can be verified similarly. \square

Notice that if \mathcal{P} is bounded then N from the proof above is the bound on the depth of recursion. Since the size of the constructed automaton is exponential in the size of the program \mathcal{P} , the UCQ which is equivalent to this program consists of proof trees of size at most exponential in the size of \mathcal{P} .

5 Boundedness vs Equivalence

In this section we focus on the similarities between the boundedness and the equivalence problem for datalog programs. In Sects. 3 and 4 those problems are treated separately but with similar techniques. Also in [13], where boundedness and equivalence are considered for monadic programs on arbitrary structures, both problems are solved using the same automata-theoretic construction. For these reasons we investigate the connection between the two problems in more detail. In contrast to the previous sections, in this section the structures under consideration are not necessarily trees or words.

Definition 1. *A class \mathcal{C} of datalog programs over a fixed class of databases is called **well-behaved** if:*

1. For every $\mathcal{P} \in \mathcal{C}$ all UCQs corresponding to the proof trees for \mathcal{P} belong to \mathcal{C} ;
2. Containment of a UCQ in a datalog program is decidable for \mathcal{C} .

Condition (1) is satisfied for most natural classes of programs. In particular by the class of all datalog programs on arbitrary structures and the class $\text{Datalog}(\downarrow)$ on trees. For the class of datalog programs on arbitrary structures Condition (2) is also known to hold true (see [10, 14, 21]). Lemma 1 shows that the class $\text{Datalog}(\downarrow)$ on trees satisfies Condition (2). Hence both those classes are well-behaved. On the other side there are classes for which Condition (2) is not true. For example if we allow for data inequality in the signature then the containment problem can be undecidable even if both programs are UCQs (see [5]).

We say that \mathcal{C} has a **computable bound** if there exists a computable function f such that if a datalog program \mathcal{P} in \mathcal{C} is bounded and $f(\mathcal{P}) = n$ then $\mathcal{P}(D) = \mathcal{P}^n(D)$ for any database D , i.e., for bounded programs the function f returns a bound on the depth of recursion. For programs which are not bounded f returns some arbitrary natural numbers.

Example 5. It follows from the results of [13] that the class of monadic datalog programs on arbitrary structures has a computable bound. It is not stated explicitly but a closer analysis of the proofs shows that this bound is polynomial in the size of the automaton used to check if a program \mathcal{P} is bounded. For example, for a linear connected program the size of such an automaton is exponential in the size of the program.

The following theorem for a well-behaved class \mathcal{C} with a computable bound establishes a connection between the problems of boundedness and equivalence to a given UCQ.

Theorem 7. *For any well-behaved class \mathcal{C} with a computable bound the following conditions are equivalent:*

1. Boundedness is decidable,
2. It is decidable whether two programs are equivalent, given that one of them is a UCQ.

Proof. Let f be the computable bound for \mathcal{C} . For the implication from (1) to (2), let $\mathcal{P}, \mathcal{Q} \in \mathcal{C}$ and assume that \mathcal{Q} is a UCQ. Since \mathcal{C} is well-behaved, we only need to show how to decide if $\mathcal{P} \subseteq \mathcal{Q}$. If \mathcal{P} is bounded, let $n = f(\mathcal{P})$. Then \mathcal{P} is equivalent to the UCQ \mathcal{P}' that corresponds to the proof trees for \mathcal{P} of height at most n . Since UCQ containment is decidable, the claim follows.

Suppose now that \mathcal{P} is not bounded. Even then \mathcal{P} can be equivalent to \mathcal{Q} (see Example 4). Let \mathcal{R} be a program containing the rules of both programs \mathcal{P} and \mathcal{Q} . We can assume that the intensional predicates in \mathcal{P} and \mathcal{Q} are all different. We add two rules to \mathcal{R} : $\mathcal{R}(X) \leftarrow \mathcal{P}(X)$ and $\mathcal{R}(X) \leftarrow \mathcal{Q}(X)$. The goal predicate \mathcal{R} holds for X iff we have $\mathcal{P}(X)$ or $\mathcal{Q}(X)$. The atoms $\mathcal{Q}(X)$ are all inferred in one step. So, if \mathcal{R} is unbounded then there exists X satisfying $\mathcal{P}(X)$ such that $\mathcal{Q}(X)$ does not hold, and hence \mathcal{P} is not contained in \mathcal{Q} . If \mathcal{R} is bounded then using f we construct an equivalent UCQ \mathcal{R}' and check whether it is equivalent to \mathcal{Q} . If this is the case then \mathcal{P} is contained in \mathcal{Q} . Otherwise it is not.

For the other implication, consider a datalog program $\mathcal{P} \in \mathcal{C}$ and let $f(\mathcal{P}) = n$. Then \mathcal{P} is bounded iff $\mathcal{P}(D) = \mathcal{P}^n(D)$ for any database D . Let \mathcal{Q} be the UCQ that corresponds to the proof trees of \mathcal{P} of height at most n . It suffices to decide whether the programs \mathcal{P} and \mathcal{Q} are equivalent. But this is decidable from the assumption that \mathcal{C} is well-behaved. \square

While assuming that a class of programs is well-behaved is natural, the existence of a computable bound is a strong assumption. It is needed since an algorithm that solves the boundedness problem might not be constructive, meaning that we do not know how big the equivalent UCQ is. However, deciding if such a function exists is usually as hard as solving the boundedness problem. From Example 5 we know that monadic programs on arbitrary structures have a computable bound. On the other hand, the undecidability results of the boundedness problem for datalog on arbitrary structures rely heavily on the fact that such a computable bound does not exist. In [16, 17] the authors present reductions from the halting problem for 2-counter machines and Turing machines. If a datalog program is bounded then the size of the equivalent UCQ corresponds to the length of an accepting run of these machines, which of course cannot be bounded by a computable function. The results of our paper are, in this sense, similar: the positive results provide computable bounds whereas the negative results rely on the fact that such a function does not exist. For these reasons we conjecture that for well-behaved classes of datalog programs the decidability of the boundedness problem is equivalent to the decidability of finding a computable bound.

6 Conclusions

The equivalence to a given nonrecursive program and the boundedness problem for $\text{Datalog}(\downarrow, \downarrow_+)$ are undecidable. To regain decidability we considered programs that do not use the \downarrow_+ relation. We showed that equivalence to a given UCQ over ranked trees is decidable, and over unranked trees it is decidable in the case of linear programs. We also showed the decidability of boundedness on words and ranked trees. We leave open the questions of decidability of boundedness and equivalence to a UCQ of non-linear $\text{Datalog}(\downarrow)$ programs over unranked trees.

We also investigated the connection between the boundedness and the equivalence to a UCQ. We showed that these problems are equivalently decidable for classes of programs with a computable bound. We suspect, however, that the existence of a computable bound for a class of programs is equivalent to the decidability of the boundedness problem. We also leave this as an open problem.

Acknowledgments. We are grateful to Anca Muscholl, Pierre Bourhis and Filip Murlak for helpful discussions and constructive comments that lead to a great improvement of the presentation of this paper. The first and third authors were supported by Poland's National Science Center grant 2013/09/N/ST6/01170, the second author was supported by Poland's National Science Center grant 2013/11/D/ST6/03075.

References

1. Abiteboul, S., Bourhis, P., Muscholl, A., Wu, Z.: Recursive queries on trees and data trees. In: ICDT, pp. 93–104 (2013)
2. Abiteboul, S., Hull, R., Vianu, V.: *Foundations of Databases*. Addison Wesley, Boston (1995)
3. Bancilhon, F., Ramakrishnan, R.: An amateur’s introduction to recursive query processing strategies. In: ACM SIGMOD, pp. 16–52 (1986)
4. Benedikt, Michael, Bourhis, Pierre, Senellart, Pierre: Monadic Datalog Containment. In: Czumaj, Artur, Mehlhorn, Kurt, Pitts, Andrew, Wattenhofer, Roger (eds.) ICALP 2012, Part II. LNCS, vol. 7392, pp. 79–91. Springer, Heidelberg (2012)
5. Björklund, Henrik, Martens, Wim, Schwentick, Thomas: Optimizing Conjunctive Queries over Trees Using Schema Information. In: Ochmański, Edward, Tyszkiewicz, Jerzy (eds.) MFCS 2008. LNCS, vol. 5162, pp. 132–143. Springer, Heidelberg (2008)
6. Bojańczyk, Mikołaj, Murlak, Filip, Witkowski, Adam: Containment of Monadic Datalog Programs via Bounded Clique-Width. In: Halldórsson, Magnús M., Iwama, Kazuo, Kobayashi, Naoki, Speckmann, Bettina (eds.) ICALP 2015. LNCS, vol. 9135, pp. 427–439. Springer, Heidelberg (2015)
7. Bonatti, P.: On the decidability of containment of recursive datalog queries - preliminary report. In: PODS, pp. 297–306 (2004)
8. Calvanese, D., De Giacomo, G., Vardi, M.: Decidable containment of recursive queries. *Theor. Comput. Sci.* **336**(1), 33–56 (2005)
9. Ceri, S., Gottlob, G., Tanca, L.: *Logic Programming and Databases*. Springer-Verlag, New York Inc (1990)
10. Chandra, A., Lewis, H., Makowsky, J.: Embedded implicational dependencies and their inference problem. In: STOC, pp. 342–354 (1981)
11. Chandra, A., Merlin, P.: Optimal implementation of conjunctive queries in relational data bases. In: STOC, pp. 77–90 (1977)
12. Chaudhuri, S., Vardi, M.: On the equivalence of recursive and nonrecursive datalog programs. In: PODS, pp. 55–66 (1992)
13. Cosmadakis, S., Gaifman, H., Kanellakis, P., Vardi, M.: Decidable optimization problems for database logic programs (preliminary report). In: STOC, pp. 477–490 (1988)
14. Cosmadakis, S., Kanellakis, P.: Parallel evaluation of recursive rule queries. In: PODS, pp. 280–293 (1986)
15. Frochaux, A., Grohe, M., Schweikardt, N.: Monadic datalog containment on trees. In: Proceedings of the 8th Alberto Mendelzon Workshop on Foundations of Data Management (2014)
16. Gaifman, H., Mairson, H., Sagiv, Y., Vardi, M.: Undecidable optimization problems for database logic programs. *J. ACM* **40**(3), 683–713 (1993)
17. Hillebrand, G., Kanellakis, P., Mairson, H., Vardi, M.: Undecidable boundedness problems for datalog programs. *J. Log. Program.* **25**(2), 163–190 (1995)
18. Mazowiecki, Filip, Murlak, Filip, Witkowski, Adam: Monadic Datalog and Regular Tree Pattern Queries. In: Csuhaj-Varjú, Erzsébet, Dietzfelbinger, Martin, Ésik, Zoltán (eds.) MFCS 2014, Part I. LNCS, vol. 8634, pp. 426–437. Springer, Heidelberg (2014)
19. Naughton, J.: Data independent recursion in deductive databases. *J. Comput. Syst. Sci.* **38**(2), 259–289 (1989)

20. Naughton, J., Sagiv, Y.: A simple characterization of uniform boundedness for a class of recursions. *J. Log. Program.* **10**, 232–253 (1991)
21. Sagiv, Y.: Optimizing datalog programs. In: Minker, J. (ed.) *Foundations of Deductive Databases and Logic Programming*, pp. 659–698. Morgan Kaufmann, Los Altos (1988)
22. Shmueli, O.: Equivalence of datalog queries is undecidable. *J. Log. Program.* **15**(3), 231–241 (1993)
23. Vardi, M.: The complexity of relational query languages (extended abstract). In: *STOC*, pp. 137–146 (1982)

Computability on the Countable Ordinals and the Hausdorff-Kuratowski Theorem (Extended Abstract)

Arno Pauly^(✉)

Clare College, University of Cambridge, Cambridge, UK
Arno.Pauly@cl.cam.ac.uk

Abstract. While there is a well-established notion of what a computable ordinal is, the question which functions on the countable ordinals ought to be computable has received less attention so far. In order to remedy this, we explore various potential representations of the set of countable ordinals. An equivalence class of representations is then suggested as a standard, as it offers the desired closure properties. With a decent notion of computability on the space of countable ordinals in place, we can then state and prove a computable uniform version of the Hausdorff-Kuratowski theorem.

1 Introduction

In Turing's seminal paper [34], he suggested to call a real number *computable* iff its decimal expansion is. However, in the corrections [35], he pointed out that it is better to use the definition that a real number is computable, iff there is a computable sequence of rational intervals collapsing to it (an idea by Brouwer). Both definitions yield the same class of real numbers – but the natural notions of what a computable function on the real numbers that come along with them differ. For example, $x \mapsto 3x$ is only computable regarding the latter, but not the former notion.

We shall show that there is a similar phenomenon regarding the notion of a computable ordinal: While there is a very well-established notion of what a computable ordinal is, various equivalent definitions do yield different notions of what a computable function on the countable ordinal is. Like multiplication with 3 for the real numbers, some simple functions such as the maximum of two ordinals fail to be computable w.r.t. several common representations of the ordinals; whereas others do yield nice effective closure properties. We will investigate some candidates, and suggest one equivalence class of representations as the standard to be adopted.

As an application, we continue a research programme to investigate concepts from descriptive set theory in the very general setting of represented spaces, and in a fashion that produces both classical and effective results simultaneously.

A full version is available as [28].

A survey of this approach is given in [27]. One of the first theorems studied in this way is the Jayne-Rogers theorem [16] (simplified proof in [22]); a computable version holding also in some non-Hausdorff spaces was proven by the author and de Brecht in [30] using results about Weihrauch reducibility in [1]. Our goal here is to state and prove a corresponding version of the Hausdorff-Kuratowski theorem.

1.1 Represented Spaces

We shall briefly introduce the notion of a *represented space*, which underlies computable analysis [37]. For a more detailed presentation we refer to [26]. A represented space is a pair $\mathbf{X} = (X, \delta_X)$ of a set X and a partial surjection $\delta_X : \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow X$ (the representation). A represented space is called *complete*, iff its representation is a total function.

A multi-valued function between represented spaces is a multi-valued function between the underlying sets. For $f : \subseteq \mathbf{X} \rightrightarrows \mathbf{Y}$ and $F : \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \mathbb{N}^{\mathbb{N}}$, we call F a realizer of f (notation $F \vdash f$), iff $\delta_Y(F(p)) \in f(\delta_X(p))$ for all $p \in \text{dom}(f \delta_X)$.

$$\begin{array}{ccc}
 \mathbb{N}^{\mathbb{N}} & \xrightarrow{F} & \mathbb{N}^{\mathbb{N}} \\
 \downarrow \delta_{\mathbf{X}} & & \downarrow \delta_{\mathbf{Y}} \\
 \mathbf{X} & \xrightarrow{f} & \mathbf{Y}
 \end{array}$$

A map between represented spaces is called *computable (continuous)*, iff it has a computable (continuous) realizer. Similarly, we call a point $x \in \mathbf{X}$ *computable*, iff there is some computable $p \in \mathbb{N}^{\mathbb{N}}$ with $\delta_X(p) = x$. We write $\mathbf{X} \cong \mathbf{Y}$ to denote that \mathbf{X} and \mathbf{Y} are computably isomorphic.

Given two represented spaces \mathbf{X}, \mathbf{Y} we obtain a third represented space $\mathcal{C}(\mathbf{X}, \mathbf{Y})$ of functions from X to Y by letting $0^n 1 p$ be a $[\delta_X \rightarrow \delta_Y]$ -name for f , if the n -th Turing machine equipped with the oracle p computes a realizer for f . As a consequence of the UTM theorem, $\mathcal{C}(-, -)$ is the exponential in the category of continuous maps between represented spaces, and the evaluation map is even computable (as are the other canonic maps, e.g. currying).

Based on the function space construction, we can obtain the hyperspaces of open \mathcal{O} , closed \mathcal{A} , overt \mathcal{V} and compact \mathcal{K} subsets of a given represented space using the ideas of synthetic topology [8].

Let $\Delta : \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \mathbb{N}^{\mathbb{N}}$ be defined on the sequences containing only finitely many 0s, and let it map those to their tail starting immediately after the last 0, with each entry reduced by 1. This is a surjection. Given a represented space $\mathbf{X} = (X, \delta_X)$, we define the represented space $\mathbf{X}^{\nabla} := (X, \delta_X \circ \Delta)$. Informally, in this space, finitely many mindchanges are allowed. The operation ∇ even extends to an endofunctor on the category of represented spaces [29, 39].

1.2 Weihrauch Reducibility

Several of our results are negative, i.e. show that certain operations are not computable. We prefer to be more precise, and not to merely state failure of

computability. Instead, we give lower bounds for Weihrauch reducibility. The reader not interested in distinguishing degrees of non-computability may skip the remainder of the subsection, and in the rest of the paper, read any statement involving Weihrauch reducibility (\leq_W , \equiv_W , $<_W$) as merely indicating the non-computability of the maps involved.

Definition 1 (Weihrauch Reducibility). *Let f, g be multi-valued functions on represented spaces. Then f is said to be Weihrauch reducible to g , in symbols $f \leq_W g$, if there are computable functions $K, H : \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \mathbb{N}^{\mathbb{N}}$ such that $K \langle \text{id}, GH \rangle \vdash f$ for all $G \vdash g$.*

The relation \leq_W is reflexive and transitive. We use \equiv_W to denote equivalence regarding \leq_W , and by $<_W$ we denote strict reducibility. By \mathfrak{W} we refer to the partially ordered set of equivalence classes. As shown in [3, 25], \mathfrak{W} is a distributive lattice. The algebraic structure on \mathfrak{W} has been investigated in further detail in [5, 15].

A prototypic non-computable function is $\text{LPO} : \mathbb{N}^{\mathbb{N}} \rightarrow \{0, 1\}$ defined via $\text{LPO}(0^{\mathbb{N}}) = 1$ and $\text{LPO}(p) = 0$ for $p \neq 0^{\mathbb{N}}$. The degree of this function was already studied by Weihrauch [36].

A few years ago several authors (Gherardi and Marcone [9], P. [24, 25], Brattka and Gherardi [2]) noticed that Weihrauch reducibility would provide a very interesting setting for a metamathematical inquiry into the computational content of mathematical theorems. The fundamental research programme was outlined in [2], and the introduction in [4] may serve as a recent survey.

2 Representations of the Space of Countable Ordinals

We shall investigate several representations of the set of all countable ordinals (to be denoted by COrd), and identify their equivalence classes up to computable translations. Along the way, we shall see how the representations of the countable ordinals restrict to the finite ordinals, and compare to established representations of the natural numbers. Theorem 3 will establish a number of candidates as equivalent, and we shall tentatively propose to consider these the standard representations of COrd . An investigation of which operations on the countable ordinals are computable is postponed until Sect. 3.

Our first candidate is a straightforward adaption of Kleene’s notation [18] of the recursive ordinals to a representation of the countable ordinals. Here and below we use a countable standard pairing function $\langle \cdot, \cdot \rangle : (\mathbb{N}^{\mathbb{N}})^{\mathbb{N}} \rightarrow \mathbb{N}^{\mathbb{N}}$.

Definition 2. *We define $\delta_K : \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \text{COrd}$ inductively via:*

1. $\delta_K(0p) = 0$
2. $\delta_K(1p) = \delta_K(p) + 1$
3. $\delta_K(2\langle p_0, p_1, p_2, \dots \rangle) = \sup_{i \in \mathbb{N}} \delta_K(p_i)$, provided that $\forall i \in \mathbb{N} \delta_K(p_i) < \delta_K(p_{i+1})$.

A potential modification of the preceding definition that immediately comes to mind would be to drop the restriction of sup’s to increasing sequences. We thus arrive at:

Definition 3. We define $\delta_{nK} : \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \text{COrd}$ inductively via:

1. $\delta_{nK}(0p) = 0$
2. $\delta_{nK}(1p) = \delta_{nK}(p) + 1$
3. $\delta_{nK}(2\langle p_0, p_1, p_2, \dots \rangle) = \sup_{i \in \mathbb{N}} \delta_{nK}(p_i)$.

A third definition proceeding along similar lines can be extracted from Moschovakis' definition of the Borel codes in [21]:

Definition 4. We define $\delta_M : \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \text{COrd}$ inductively via:

1. $\delta_M(0p) = 0$
2. $\delta_M(1\langle p_0, p_1, p_2, \dots \rangle) = \sup_{i \in \mathbb{N}} (\delta_M(p_i) + 1)$.

Another scheme to obtain representations of the countable ordinals starts with the view of countable ordinals as the heights of countable wellfounded relations. A countable relation is given by two sets $A \subseteq \mathbb{N}$ and $R \subseteq \mathbb{N} \times \mathbb{N}$, where A denotes which points are present, and then R provides the order relation. There are three common spaces of subsets of \mathbb{N} , the open subsets $\mathcal{O}(\mathbb{N})$, the closed subsets $\mathcal{A}(\mathbb{N})$ or the clopens $\mathcal{O}(\mathbb{N}) \wedge \mathcal{A}(\mathbb{N})$. The computable points in these spaces are the recursively enumerable, the co-recursively enumerable and the decidable subsets of \mathbb{N} respectively. Thus, we arrive at a number of representations:

Definition 5. Let $X, Y \in \{\mathcal{O}(\mathbb{N}), \mathcal{A}(\mathbb{N}), \mathcal{O}(\mathbb{N}) \wedge \mathcal{A}(\mathbb{N})\}$. We define a representation $\delta_R^{X,Y} : \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \text{COrd}$ by $\delta_R^{X,Y}(\langle p, q \rangle) = \alpha$, iff α is the height of the poset $(A, <)$, where p is an X -name for A , q an Y -name for R , and $\forall i, j \in A (i < j \Leftrightarrow \langle i, j \rangle \in R)$.

Potentially, it would appear to be more appropriate to consider countable ordinals as order types of countable wellorders, rather than just heights of wellfounded orders. This is the approach taken by Hamkins and Li [20].

Definition 6. Let $X, Y \in \{\mathcal{O}(\mathbb{N}), \mathcal{A}(\mathbb{N}), \mathcal{O}(\mathbb{N}) \wedge \mathcal{A}(\mathbb{N})\}$. Let $\delta_{wR}^{X,Y} : \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \text{COrd}$ be the restriction of $\delta_R^{X,Y}$ to those $\langle p, q \rangle$ where q encodes a wellorder.

Finally, we introduce a representation tailor-made for the formulation and proof of a computable Hausdorff-Kuratowski theorem below. Let a nice relation be a well-founded quasi-order \preceq on \mathbb{N} , such that $\forall n, n \preceq 0$, and whenever $n < m$, then $n > m$.

Definition 7. We define a representation $\delta_{nR} : \subseteq \{0, 1\}^{\mathbb{N}} \rightarrow \text{COrd}$ by $\delta_{nR}(p) = \alpha$, iff the relation \preceq_p defined via $n \preceq_p m$ iff $p(\langle n, m \rangle) = 1$ is a nice relation of height $\alpha + 1$ (the height of any nice relation is a countable successor ordinal, and every countable successor ordinal arises as the height of some nice relation).

To obtain some initial understanding of how the various representations work, we shall consider what happens to the finite ordinals. Besides the usual natural numbers \mathbb{N} , also the spaces $\mathbb{N}_{<}$, $\mathbb{N}_{>}$ and \mathbb{N}^{∇} , where a number n is represented by a non-decreasing, respectively non-increasing, respectively arbitrary sequence of integers which eventually converge to n .

Observation 1. $(\text{id} : \mathbb{N}^{\nabla} \rightarrow \mathbb{N}) \equiv_W (\text{id} : \mathbb{N}_{<} \rightarrow \mathbb{N}) \equiv_W C_{\mathbb{N}}$; $(\text{id} : \mathbb{N}_{>} \rightarrow \mathbb{N}) \equiv_W LPO^*$ and $LPO \leq_W (\text{id} : \mathbb{N}_{>} \rightarrow \mathbb{N}_{<})$.

Proposition 1.

1. $(COrd, \delta_K) \upharpoonright_{\mathbb{N}} \cong \mathbb{N}$
2. $(COrd, \delta_{nK}) \upharpoonright_{\mathbb{N}} \cong \mathbb{N}_{<}$
3. $(COrd, \delta_M) \upharpoonright_{\{n \in \mathbb{N} \mid n > 0\}} \cong (\mathbb{N}_{<}) \upharpoonright_{\{n \in \mathbb{N} \mid n > 0\}}$
4. $(COrd, \delta_{wR}^{\mathcal{A}(\mathbb{N}), Y}) \upharpoonright_{\mathbb{N}} \cong (COrd, \delta_R^{\mathcal{A}(\mathbb{N}), Y}) \upharpoonright_{\mathbb{N}} \cong \mathbb{N}^\nabla$ (regardless of the choice of $Y \in \{\mathcal{O}(\mathbb{N}), \mathcal{A}(\mathbb{N}), \mathcal{O}(\mathbb{N}) \wedge \mathcal{A}(\mathbb{N})\}$)
5. $(COrd, \delta_{wR}^{\mathcal{O}(\mathbb{N}), \mathcal{O}(\mathbb{N})}) \upharpoonright_{\mathbb{N}} \cong \mathbb{N}_{<}$

We can extend Proposition 1 (3) to:

Lemma 1. $(COrd, \delta_M) \upharpoonright_{\{\alpha > 0\}} \cong (COrd, \delta_{nK}) \upharpoonright_{\{\alpha > 0\}}$

Merely requiring the domain of the structure to be enumerable, rather than decidable, does not impact the representation at all though. For not necessarily wellordered relations, the same applies to the relation itself.

Lemma 2. $(COrd, \delta_R^{\mathcal{O}(\mathbb{N}), Y}) \cong (COrd, \delta_R^{\mathcal{O}(\mathbb{N}) \wedge \mathcal{A}(\mathbb{N}), Y})$ and $(COrd, \delta_{wR}^{\mathcal{O}(\mathbb{N}), Y}) \cong (COrd, \delta_{wR}^{\mathcal{O}(\mathbb{N}) \wedge \mathcal{A}(\mathbb{N}), Y})$

Lemma 3. $(COrd, \delta_R^{X, \mathcal{O}(\mathbb{N})}) \cong (COrd, \delta_R^{X, \mathcal{O}(\mathbb{N}) \wedge \mathcal{A}(\mathbb{N})})$

Proof. Essentially, whenever new information about the relationship between two already settled points occurs, one can create a fresh copy of everything encountered so far. As smaller relations (w.r.t subset inclusion) have smaller height, the extra copy does not impact the ordinal represented thus. \square

Theorem 1. Let δ be a representation of $COrd$ such that

1. $0 \in (COrd, \delta)$
2. $+1 : (COrd, \delta) \rightarrow (COrd, \delta)$
3. $\text{sup} : \mathcal{C}(\mathbb{N}, (COrd, \delta)) \rightarrow (COrd, \delta)$

are all computable. Then $\text{id} : (COrd, \delta_{nK}) \rightarrow (COrd, \delta)$ is computable.

Proof. Induction along the definition of δ_{nK} . \square

Theorem 2. The following representations are equivalent:

$$\delta_{nK} \qquad \delta_{nR} \qquad \delta_R^{\mathcal{O}(\mathbb{N}) \wedge \mathcal{A}(\mathbb{N}), \mathcal{O}(\mathbb{N}) \wedge \mathcal{A}(\mathbb{N})}$$

$$\delta_R^{\mathcal{O}(\mathbb{N}), \mathcal{O}(\mathbb{N}) \wedge \mathcal{A}(\mathbb{N})} \quad \delta_R^{\mathcal{O}(\mathbb{N}) \wedge \mathcal{A}(\mathbb{N}), \mathcal{O}(\mathbb{N})} \quad \delta_R^{\mathcal{O}(\mathbb{N}), \mathcal{O}(\mathbb{N})}$$

Theorem 3.

1. $(\text{id} : (COrd, \delta_K) \rightarrow (COrd, \delta_{nK}))$ is computable, but $C_{\mathbb{N}} \leq_W (\text{id} : (COrd, \delta_{nK}) \rightarrow (COrd, \delta_K))$

2. $(\text{id} : (\mathbf{COrd}, \delta_M) \rightarrow (\mathbf{COrd}, \delta_{nK}))$ is computable,
but $LPO \equiv_W (\text{id} : (\mathbf{COrd}, \delta_{nK}) \rightarrow (\mathbf{COrd}, \delta_M))$
3. $(\text{id} : (\mathbf{COrd}, \delta_{nK}) \rightarrow (\mathbf{COrd}, \delta_R^{X, \mathcal{A}(\mathbb{N})}))$ is computable,
but $LPO^* \leq_W (\text{id} : (\mathbf{COrd}, \delta_R^{X, \mathcal{A}(\mathbb{N})}) \rightarrow (\mathbf{COrd}, \delta_K))$
4. $(\text{id} : (\mathbf{COrd}, \delta_{nK}) \rightarrow (\mathbf{COrd}, \delta_R^{\mathcal{A}(\mathbb{N}), Y}))$ is computable,
but $LPO^* \leq_W (\text{id} : (\mathbf{COrd}, \delta_R^{\mathcal{A}(\mathbb{N}), Y}) \rightarrow (\mathbf{COrd}, \delta_K))$
5. $(\text{id} : (\mathbf{COrd}, \delta_{wR}^{\mathcal{O}, \mathcal{O}}) \rightarrow (\mathbf{COrd}, \delta_{nK}))$ is computable,
but $(\text{id} : (\mathbf{COrd}, \delta_{nK}) \rightarrow (\mathbf{COrd}, \delta_{wR}^{\mathcal{A} \wedge \mathcal{O}, \mathcal{A} \wedge \mathcal{O}}))$ is not computable.
6. $(\text{id} : (\mathbf{COrd}, \delta_K) \rightarrow (\mathbf{COrd}, \delta_M))$ is computable,
but $C_{\mathbb{N}} \leq_W (\text{id} : (\mathbf{COrd}, \delta_M) \rightarrow (\mathbf{COrd}, \delta_K))$

Definition 8. We will consider the equivalence class of δ_{nK} identified in Theorem 2 as the standard representation of \mathbf{COrd} , and thus abbreviate $\mathbf{COrd} := (\mathbf{COrd}, \delta_{nK})$.

Besides \mathbf{COrd} , we will also consider $\mathbf{COrd}_M := (\mathbf{COrd}, \delta_M)$, $\mathbf{COrd}_K := (\mathbf{COrd}, \delta_K)$ and $\mathbf{COrd}_{HL} := (\mathbf{COrd}, \delta_{wR}^{\mathcal{A} \wedge \mathcal{O}, \mathcal{A} \wedge \mathcal{O}})$. Their mutual relations are demonstrated in Fig. 1. The representations using well-founded structures given as closed sets would seem to be too weak to be of much interest, and thus will no longer be considered.

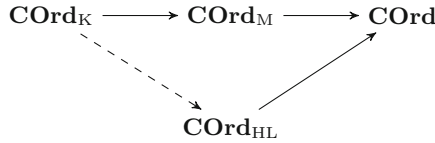


Fig. 1. Translatability between the representations. The dashed arrow refers to the Open Question 2

3 Computability on \mathbf{COrd}

In order to justify the stance that the represented space \mathbf{COrd} really is *the* space of countable ordinals, we shall investigate the computable operations on it and related properties.

Theorem 4. *The following operations are computable:*

1. $+$: $\mathbf{COrd} \times \mathbf{COrd} \rightarrow \mathbf{COrd}$
2. \times : $\mathbf{COrd} \times \mathbf{COrd} \rightarrow \mathbf{COrd}$
3. sup : $\mathbf{COrd}^{\mathbb{N}} \rightarrow \mathbf{COrd}$
4. (-1) : $\mathbf{COrd} \rightarrow \mathbf{COrd}$, where $(-1)(\alpha + 1) = \alpha$ and for limit ordinals γ , $(-1)(\gamma) = \gamma$

- 5. Smaller : $\mathbf{COrd} \rightrightarrows \mathbf{COrd}^{\mathbb{N}}$ where $(\alpha_i)_{i \in \mathbb{N}} \in \text{Smaller}(\alpha)$ iff $\{0\} \cup \{\beta \in \mathbf{COrd} \mid \beta < \alpha\} = \{\alpha_i \mid i \in \mathbb{N}\}$
- 6. $(\alpha, \beta) \mapsto \alpha^\beta : \mathbf{COrd} \times \mathbf{COrd} \rightarrow \mathbf{COrd}$

Proposition 2. $LPO^* \leq_W (- : \mathbf{COrd} \times \mathbf{COrd} \rightarrow \mathbf{COrd})$

4 Computability on \mathbf{COrd}_K

In order to define the concept of a *computable ordinal*, Kleene’s definition resulting in the space \mathbf{COrd}_K seems to be the typical choice. A strong reason to reject \mathbf{COrd}_K as the natural candidate for computability on the countable ordinals nonetheless, lies in the following result:

Proposition 3. $LPO \leq_W (\max : \mathbf{COrd}_K \times \mathbf{COrd}_K \rightarrow \mathbf{COrd}_K)$

The reason that calling the computable elements in \mathbf{COrd}_K the computable ordinals is justified regardless of \mathbf{COrd}_K not being the right space lies in the fact that both \mathbf{COrd}_K and \mathbf{COrd} have the same computable points. This situation is somewhat reminiscent of Turing’s transient mistake of defining the computable real numbers via the decimal expansion at first [34] before correcting himself [35].

Proposition 4. *The map $\text{UpperBound} : \mathbf{COrd} \rightrightarrows \mathbf{COrd}_K$ defined by $\beta \in \text{UpperBound}(\alpha)$ iff $\beta \geq \alpha$ is computable.*

Proof. The computation proceeds by induction, using the representations δ_{nK} and δ_K . For 0 and successor, both representations agree anyway. Given a supremum $\alpha = \sup_{n \in \mathbb{N}} \alpha_n$, we apply UpperBound to each α_n to obtain an upper bound β_n . Now $\beta = \sup_{n \in \mathbb{N}} (\beta_0 + \dots + \beta_n)$ is a valid output for $\text{UpperBound}(\alpha)$ (note that addition is computable on \mathbf{COrd}_K). □

Corollary 1. *The computable elements of \mathbf{COrd}_K , \mathbf{COrd}_M and \mathbf{COrd} are the same.*

Proof. From Proposition 4 in conjunction with Theorem 4 (5). □

5 \mathbf{COrd}_M and Boundedness

Given that \mathbf{COrd}_M is very similar to \mathbf{COrd} , only differing in the properties of 0, and that \mathbf{COrd} has the better closure properties (as sup is not computable on \mathbf{COrd}_M^1), one may wonder what the point of this space is. The special treatment of 0 in \mathbf{COrd}_M allows us to obtain a very useful extension of the \leq -relation on \mathbf{COrd}_M , which ultimately can be used to prove that all continuous functions from Baire space into the countable ordinals are bounded:

¹ Any algorithm attempting to compute sup on \mathbf{COrd}_M needs to decide whether or not the result is 0 after finitely many steps – and this questions essentially is LPO.

Theorem 5 (Gregoriades, Kispéter and P. [10]²). *For every continuous (even: every Borel-measurable) function $f : \mathbb{N}^{\mathbb{N}} \rightarrow \mathbf{COrd}_M$ there is some $\alpha \in \mathbf{COrd}$ such that $\forall p \in \mathbb{N}^{\mathbb{N}} \quad f(p) \leq \alpha$.*

Corollary 2. *For every continuous (even: every Borel-measurable) function $f : \mathbb{N}^{\mathbb{N}} \rightarrow \mathbf{COrd}$ there is some $\alpha \in \mathbf{COrd}$ such that $\forall p \in \mathbb{N}^{\mathbb{N}} \quad f(p) \leq \alpha$.*

Proof. Using Theorem 5 together with Proposition 4. □

Corollary 3. *There is no total representation $\delta : \mathbb{N}^{\mathbb{N}} \rightarrow \mathbf{COrd}$ such that $\text{id} : (\mathbf{COrd}, \delta) \rightarrow \mathbf{COrd}$ could be Borel measurable.*

Unfortunately, the proof of Theorem 5 is entirely non-constructive and does not offer a way to extract a bound from a description of the function. As a result of Spector establishes the corresponding version in the computable discrete realm, there seems to be hope for a positive answer to at least the weak version of the following:

Question 1. Is the function $\text{sup} : \mathcal{C}(\mathbb{N}^{\mathbb{N}}, \mathbf{COrd}) \rightarrow \mathbf{COrd}$ computable? Is the multifunction $\text{UpperBound} : \mathcal{C}(\mathbb{N}^{\mathbb{N}}, \mathbf{COrd}) \rightrightarrows \mathbf{COrd}$ computable?

6 Computability on \mathbf{COrd}_{HL}

Computability on the space \mathbf{COrd}_{HL} was studied by Joel Hamkins and Zhenhao Li in [20]. We briefly survey some of their results:

Theorem 6 (Hamkins and Li [20]). *The following operations are computable:*

1. $+$: $\mathbf{COrd}_{HL} \times \mathbf{COrd}_{HL} \rightarrow \mathbf{COrd}_{HL}$
2. \times : $\mathbf{COrd}_{HL} \times \mathbf{COrd}_{HL} \rightarrow \mathbf{COrd}_{HL}$
3. $(\alpha, \beta) \mapsto \alpha^\beta$: $\mathbf{COrd}_{HL} \times \mathbf{COrd}_{HL} \rightarrow \mathbf{COrd}_{HL}$
4. $\alpha + 1 \mapsto \alpha : \subseteq \mathbf{COrd}_{HL} \rightarrow \mathbf{COrd}_{HL}$
5. $\omega^{CK} + \omega \mapsto \omega^{CK} : \subseteq \mathbf{COrd}_{HL} \rightarrow \mathbf{COrd}_{HL}$

As with Proposition 3 for \mathbf{COrd}_K , the first item of the following justifies our rejection of \mathbf{COrd}_{HL} as proposed *standard computability structure* on the countable ordinals. We point out that the technique introduced in [20, Theorem 16] essentially is a Wadge game relative to the representation, similar to the generalizations of the classical Wadge hierarchy on $\mathbb{N}^{\mathbb{N}}$ to represented spaces in [31] by Pequignot and [7] by Duparc and Fournier.

Theorem 7 (Hamkins and Li [20]). *The following operations are not computable:*

1. \max : $\mathbf{COrd}_{HL} \times \mathbf{COrd}_{HL} \rightarrow \mathbf{COrd}_{HL}$
2. $\alpha \mapsto \max\{\alpha, \omega + 1\}$: $\mathbf{COrd}_{HL} \rightarrow \mathbf{COrd}_{HL}$
3. $\omega \times \alpha \mapsto \alpha : \subseteq \mathbf{COrd}_{HL} \rightarrow \mathbf{COrd}_{HL}$

² This result essentially is folklore.

- 4. $Reduce_n \subseteq \mathbf{COrd}_{HL} \rightarrow \mathbf{COrd}_{HL}$ where $Reduce_n(\omega) = n$ and $Reduce_n(\omega + \omega) = \omega$
- 5. $D \subseteq \mathbf{COrd}_{HL} \rightarrow \{0, 1\}$ where $D(\omega) = 0$ and $D(\omega + 1) = 1$

Corollary 4. $id : \mathbf{COrd}_{HL} \rightarrow \mathbf{COrd}_K$ is not computable.

An open question raised in [20] is whether the supremum of strictly increasing sequences of ordinals can be computed. This boils down to the following:

Question 2 (Hamkins and Li [20]). Is $id : \mathbf{COrd}_K \rightarrow \mathbf{COrd}_{HL}$ computable?

Finally, we point out that the investigations in [20, Sect.5] concern the *point degree spectrum* of \mathbf{COrd}_{HL} (without using this terminology, though). Point degree spectra of represented spaces were introduced by Kihara and P. in [17].

7 A Non-deceiving Representation of COrd?

The trusted recipe of identifying suitable representations of some structure is to pick an admissible representation whose final topology coincides with some natural topology on the structure³. However, the usual topology on COrd would be the order topology, which is not separable – and every represented space is separable. In this section, we shall explore whether a weaker topological requirement could be imposed on a representation.

Inspired by a property studied in the context of winning conditions for infinite sequential games in [19] by Le Roux and P., we shall call a function $f : \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \mathbf{COrd}$ *non-deceiving*, iff whenever $(p_n)_{n \in \mathbb{N}}$ is a sequence converging to p in $\text{dom}(f)$ such that $\forall n \in \mathbb{N} f(p_n) < f(p_{n+1})$, then $\forall i \in \mathbb{N} f(p_i) < f(p)$.

Theorem 8 (Gregoriades⁴). Any non-deceiving function $f : \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \mathbf{COrd}$ is bounded by some countable ordinal.

Corollary 5. There is no non-deceiving representation of COrd.

The preceding corollary presumably destroys any hope to find a suitable representation of COrd that is admissible w.r.t. some weak limit space structure in the sense of Schröder [32,33].

8 The Computable Hausdorff-Kuratowski Theorem

We shall now prepare the formulation of the Hausdorff-Kuratowski theorem in the framework of computable endofunctors on the category of represented spaces as introduced by de Brecht and P. in [6,29,30]. The setting closely follows the

³ In fact, it is sometimes claimed that it *has* to be done like that – the present work ought to disprove this.

⁴ This theorem is based on a personal communication by Vassilios Gregoriades.

corresponding section in [6] by de Brecht, where a weaker (and non-effective) version of our desired result was proven.

For any sequence of countable ordinals $(\alpha_i)_{i \in \mathbb{N}}$, we define a function $L_{(\alpha_i)_{i \in \mathbb{N}}} : \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \mathbb{N}^{\mathbb{N}}$. The sequence only impacts the domain, but whenever $L_{(\alpha_i)_{i \in \mathbb{N}}}(p)$ is defined, then $2L_{(\alpha_i)_{i \in \mathbb{N}}}(p)(n) = p(\max\{i \in \mathbb{N} \mid p(i) \text{ is odd}\} + n + 1)$; i.e. $L_{(\alpha_i)_{i \in \mathbb{N}}}$ takes the maximal tail of its input consisting of only even values, and returns the result of pointwise division by 2. Obviously any sequence in the domain of $L_{(\alpha_i)_{i \in \mathbb{N}}}$ has to contain only finitely many odd entries; and we additionally demand that for $p \in \text{dom}(L_{(\alpha_i)_{i \in \mathbb{N}}})$, if $n < m$, and $p(n) = 2k + 1$ and $p(m) = 2j + 1$, then $\alpha_k > \alpha_j$.

Definition 9. We define a computable endofunctor $\mathfrak{L}_{(\alpha_n)_{n \in \mathbb{N}}}$ by $\mathfrak{L}_{(\alpha_n)_{n \in \mathbb{N}}}(X, \delta) = (X, \delta \circ L_{(\alpha_i)_{i \in \mathbb{N}}})$ and the straightforward extension to functions.

Each endofunctor $\mathfrak{L}_{(\alpha_n)_{n \in \mathbb{N}}}$ captures a version of computability with finitely many mindchanges (e.g. [38, 39]): The regular outputs are encoded as even numbers. Finitely many times, the output can be reset by using an odd number, however, when doing so, one has to count down within the list of ordinals parameterizing the function (which in particular ensures that it happens only finitely many times). We thus find it connected to the *level* introduced by Hertling [12], and further studied by him and others in [6, 11, 13, 14, 23, 25].

Definition 10. Given a function $f : \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \mathbb{N}^{\mathbb{N}}$, we define the sets $\mathcal{L}_\alpha(f) \subseteq \mathbb{N}^{\mathbb{N}}$ inductively via:

1. $\mathcal{L}_0(f) = \text{dom}(f)$
2. $\mathcal{L}_{\alpha+1}(f) = \overline{\{x \in \mathcal{L}_\alpha(f) \mid f|_{\mathcal{L}_\alpha} \text{ is discontinuous at } x\}}$
3. $\mathcal{L}_\gamma(f) = \bigcap_{\beta < \gamma} \mathcal{L}_\beta(f)$ for limit ordinals γ .

Then we say $\text{Lev}(f) := \min\{\alpha \mid \mathcal{L}_\alpha(f) = \emptyset\}$.

Theorem 9. If $f : \mathbb{N}^{\mathbb{N}} \rightarrow \mathfrak{L}_{(\alpha_i)_{i \in \mathbb{N}}} \mathbb{N}^{\mathbb{N}}$ is continuous, then $\text{Lev}(f) \leq (\sup_{i \in \mathbb{N}} \alpha_i) + 1$.

Proposition 5. Let $(\alpha_i)_{i \in \mathbb{N}}$ be such that $\exists \alpha \in \text{COrd}$ with $\{\alpha_i \mid i \in \mathbb{N}\} = \{\beta \in \text{COrd} \mid \beta < \alpha\}$. Then $\text{Lev}(L_{(\alpha_i)_{i \in \mathbb{N}}}) = \alpha + 1$.

The computable Hausdorff-Kuratowski theorem has at its heart a dependent sum type; namely the construction $\sum_{(\alpha_i)_{i \in \mathbb{N}} \in \text{COrd}^{\mathbb{N}}} (\mathcal{C}(\mathbf{X}, \mathfrak{L}_{(\alpha_i)_{i \in \mathbb{N}}} \mathbf{Y}))$ for some represented spaces \mathbf{X}, \mathbf{Y} . A point in this space is a pair, consisting of a sequence of countable ordinals and a function $f : \mathbf{X} \rightarrow \mathbf{Y}$, the latter given only in a $\mathfrak{L}_{(\alpha_i)_{i \in \mathbb{N}}}$ -continuous way.

Theorem 10 (Computable Hausdorff-Kuratowski Theorem). Let \mathbf{X}, \mathbf{Y} be represented spaces, and \mathbf{X} be complete. Then the map $\text{HK} : \mathcal{C}(\mathbf{X}, \mathbf{Y}^\nabla) \rightrightarrows \sum_{(\alpha_i)_{i \in \mathbb{N}} \in \text{COrd}^{\mathbb{N}}} (\mathcal{C}(\mathbf{X}, \mathfrak{L}_{(\alpha_i)_{i \in \mathbb{N}}} \mathbf{Y}))$ where $((\alpha_i)_{i \in \mathbb{N}}, g) \in \text{HK}(f)$ iff $f = g$, is computable.

Corollary 6. *Let $f : \mathbf{X} \rightarrow \mathbf{Y}$ be computable with finitely many mindchanges, and \mathbf{X} be complete. Then $\text{Lev}(f)$ exists and is a computable ordinal.*

The result of the preceding corollary was also announced by Selivanov at CCA 2014.

Acknowledgements. I am grateful to Victor Selivanov for sparking my interest in a computable version of the Hausdorff Kuratowski theorem and to Vasco Brattka and Matthew de Brecht for various discussions on this question. The comparison of the various representations of the countable ordinals started with a discussion with Vassilios Gregoriades.

This work benefited from the Royal Society International Exchange Grant IE111233 and the Marie Curie International Research Staff Exchange Scheme *Computable Analysis*, PIRSES-GA-2011- 294962.

References

1. Brattka, V., de Brecht, M., Pauly, A.: Closed choice and a uniform low basis theorem. *Ann. Pure Appl. Logic* **163**(8), 968–1008 (2012)
2. Brattka, V., Gherardi, G.: Effective choice and boundedness principles in computable analysis. *Bull. Symbolic Logic* **1**, 73–117 (2011). [arXiv:0905.4685](#)
3. Brattka, V., Gherardi, G.: Weihrauch degrees, omniscience principles and weak computability. *J. Symbolic Logic* **76**, 143–176 (2011). [arXiv:0905.4679](#)
4. Brattka, V., Gherardi, G., Hölzl, R.: Probabilistic computability and choice. [arXiv:1312.7305](#) (2013)
5. Brattka, V., Pauly, A.: On the algebraic structure of Weihrauch degrees (forthcoming)
6. de Brecht, M.: Levels of discontinuity, limit-computability, and jump operators. In: Brattka, V., Diener, H., Spreen, D. (eds.) *Logic, Computation, Hierarchies*, pp. 79–108. de Gruyter, Berlin (2014). [arXiv 1312.0697](#)
7. Duparc, J., Fournier, K.: Reductions by relatively continuous relations on $\mathbb{N}^{\leq\omega}$ (unpublished notes)
8. Escardó, M.: Synthetic topology of datatypes and classical spaces. *Electron. Notes Theoret. Comput. Sci.* **87**, 21–156 (2004)
9. Gherardi, G., Marcone, A.: How incomputable is the separable Hahn-Banach theorem? *Notre Dame J. Formal Logic* **50**(4), 393–425 (2009)
10. Gregoriades, V., Kispéter, T., Pauly, A.: A comparison of concepts from computable analysis and effective descriptive set theory. [arXiv:1401.3325](#) (2014)
11. Hertling, P.: Topological complexity with continuous operations. *J. Complex.* **12**(4), 315–338 (1996)
12. Hertling, P.: *Unstetigkeitsgrade von funktionen in der effektiven analysis*. Ph.D. thesis, Fernuniversität, Gesamthochschule in Hagen, Oktober 1996
13. Hertling, P.: Topological complexity of zero finding with algebraic operations. *J. Complex.* **18**, 912–942 (2002)
14. Hertling, P., Weihrauch, K.: Levels of degeneracy and exact lower complexity bounds. In: 6th Canadian Conference on Computational Geometry, pp. 237–242 (1994)
15. Higuchi, K., Pauly, A.: The degree-structure of Weihrauch-reducibility. *Log. Methods Comput. Sci.* **9**(2), 1–17 (2013)

16. Jayne, J., Rogers, C.: First level borel functions and isomorphisms. *Journal de Mathématiques Pures et Appliquées* **61**, 177–205 (1982)
17. Kihara, T., Pauly, A.: Point degree spectra of represented spaces. [arXiv:1405.6866](https://arxiv.org/abs/1405.6866) (2014)
18. Kleene, S.C.: On notation for ordinal numbers. *J. Symbolic Logic* **3**, 150–155 (1938)
19. Le Roux, S., Pauly, A.: Infinite sequential games with real-valued payoffs [arXiv:1401.3325](https://arxiv.org/abs/1401.3325) (2014). [arXiv:1401.3325](https://arxiv.org/abs/1401.3325)
20. Li, Z., Hamkins, J.D.: On effectiveness of operations on countable ordinals. unpublished notes
21. Moschovakis, Y.N.: *Descriptive Set Theory, Studies in Logic and the Foundations of Mathematics*, vol. 100. North-Holland publishing company, Amsterdam (1980)
22. Ros, M.L., Semmes, B.: A new proof of a theorem of Jayne and Rogers. *Real Anal. Exch.* **35**(1), 195–204 (2009)
23. Pauly, A.: *Methoden zum Vergleich der Unstetigkeit von Funktionen*. Masters thesis, FernUniversität Hagen (2007)
24. Pauly, A.: How incomputable is finding Nash equilibria? *J. Univ. Comput. Sci.* **16**(18), 2686–2710 (2010)
25. Pauly, A.: On the (semi)lattices induced by continuous reducibilities. *Math. Logic Q.* **56**(5), 488–502 (2010)
26. Pauly, A.: On the topological aspects of the theory of represented spaces. [arXiv:1204.3763](https://arxiv.org/abs/1204.3763) (2012)
27. Pauly, A.: The descriptive theory of represented spaces. [arXiv:1408.5329](https://arxiv.org/abs/1408.5329) (2014)
28. Pauly, A.: Computability on the countable ordinals and the Hausdorff-Kuratowski theorem. [arXiv 1501.00386](https://arxiv.org/abs/1501.00386) (2015)
29. Pauly, A., de Brecht, M.: Towards synthetic descriptive set theory: An instantiation with represented spaces. [arXiv:1307.1850](https://arxiv.org/abs/1307.1850) (2013)
30. Pauly, A., de Brecht, M.: Non-deterministic computation and the Jayne Rogers theorem. *Electron. Proc. Theoret. Comput. Sci.* **143**, 87–96 (2014). dCM 2012
31. Pequignot, Y.: A Wadge hierarchy for second countable spaces. *Arch. Math. Logic*, 1–25 (2015). <http://dx.doi.org/10.1007/s00153-015-0434-y>
32. Schröder, M.: *Admissible Representations for Continuous Computations*. Ph.D. thesis, FernUniversität Hagen (2002)
33. Schröder, M.: A natural weak limit space with admissible representation which is not a limit space. *ENTCS* **66**(1), 165–175 (2002)
34. Turing, A.: On computable numbers, with an application to the Entscheidungsproblem. *Proc. LMS* **2**(42), 230–265 (1936)
35. Turing, A.: On computable numbers, with an application to the Entscheidungsproblem: Corrections. *Proc. LMS* **2**(43), 544–546 (1937)
36. Weihrauch, K.: *The TTE-interpretation of three hierarchies of omniscience principles*. Informatik Berichte 130, FernUniversität Hagen, Hagen (1992)
37. Weihrauch, K.: *Computable Analysis*. Springer, Heidelberg (2000)
38. Ziegler, M.: Real hypercomputation and continuity. *Theory Comput. Syst.* **41**, 177–206 (2007)
39. Ziegler, M.: Revising type-2 computation and degrees of discontinuity. *Electron. Notes Theoret. Comput. Sci.* **167**, 255–274 (2007)

Emergence on Decreasing Sandpile Models

Kévin Perrot¹(✉) and Éric Rémila²

¹ Aix Marseille Université - CNRS - LIF UMR 7279, 13288 Marseille, France

`kevin.perrot@lif.univ-mrs.fr`

² Université de Lyon - CNRS - GATE LSE UMR 5824, 42023 Saint-Etienne, France

`eric.remila@univ-st-etienne.fr`

Abstract. Sand is a proper instance for the study of natural algorithmic phenomena. Idealized square/cubic sand grains moving according to “simple” local toppling rules may exhibit surprisingly “complex” global behaviors. In this paper we explore the language made by words corresponding to fixed points reached by iterating a toppling rule starting from a finite stack of sand grains in one dimension. Using arguments from linear algebra, we give a constructive proof that for all decreasing sandpile rules the language of fixed points is accepted by a finite (Muller) automaton. The analysis is completed with a combinatorial study of cases where the *emergence* of precise regular patterns is formally proven. It extends earlier works presented in [15–17], and asks how far can we understand and explain emergence following this track?

Keywords: Sandpile models · Fixed points · Emergence

1 Introduction

In the spirit of Chazelle’s natural algorithms [3], we propose a study of sandpile models via the tools of theoretical computer science. In this discrete dynamical system (DDS), configurations are words over the alphabet \mathbb{N} (or \mathbb{Z}), where each letter is a number of stacked sand grains. The dynamics is described by local rules letting grains move from column to column. Historically, discrete sandpile models were first introduced by the physicists Bak, Tang and Wiesenfeld, as a paradigmatic example of *self-organized critical* systems in which instabilities are described by power laws [2]. The combinatorial study of one-dimensional models has been initiated by Goles in [9] as a case where simple rules lead to surprisingly complex, yet tractable, mathematical problems on term rewriting systems. A particular focus has been given to the dynamics starting from a finite pile of sand grains (which is equivalent to adding them one by one, as they would fall in an hourglass), and to the structure of the language \mathcal{L} made of fixed point configurations reached at the end of the stabilization process [5–7, 11].

This work was partially supported by IXXI (Complex System Institute, Lyon), ANR projects Dynamite and QuasiCool (ANR-12-JS02-011-01), Modmad Federation of U. St-Etienne, FONDECYT Grant 3140527 (DIM, Universidad de Chile), and Núcleo Milenio Información y Coordinación en Redes (ACGO).

The Model. Let $\mathbb{Z}_f^{\mathbb{N}}$ denote the set of integer sequences which are ultimately null (a sequence $h = (h_0, h_1, h_2, \dots)$ is ultimately null if there exists a positive integer i such that, for $j \geq i$, we have $h_j = 0$). We note by Δ the morphism $\mathbb{Z}_f^{\mathbb{N}} \rightarrow \mathbb{Z}_f^{\mathbb{N}}$ such that for any $h \in \mathbb{Z}_f^{\mathbb{N}}$ and any $i \in \mathbb{N}$, $\Delta h_i = h_i - h_{i+1}$. Notice that Δ is bijective. We say that h_i is the *number of grains* and Δh_i is the *slope of h in column i* . The set \mathcal{C} is the subset of sequences of $\mathbb{Z}_f^{\mathbb{N}}$ which are non increasing.

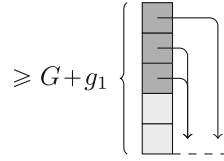


Fig. 1. The rule (2, 1) can be applied if and only if $\Delta h_i \geq 5$. For convenience, *right* is the direction of grains fall.

Definition 1. A decreasing sandpile model is a DDS defined by:

- **Configurations.** The set of configurations is the subset \mathcal{C} of $\mathbb{Z}_f^{\mathbb{N}}$,
- **Transition rule.** Determined by a p -tuple $\mathcal{R} = (g_1, g_2, \dots, g_p)$ of integers with $g_1 \geq \dots \geq g_p > 0$. From a configuration $h \in \mathcal{C}$, a transition at i leads to $h' \in \mathcal{C}$, denoted $h \xrightarrow{i} h'$, when:
 - $h'_i = h_i - G$ with $G = \sum_{j=1}^p g_j$;
 - $h'_{i+j} = h_{i+j} + g_j$ for $1 \leq j \leq p$.

To lighten them, almost all notations do not mention the rule $\mathcal{R} = (g_1, \dots, g_p)$, as it does not change during the evolution. Since configurations must be non-increasing, the rule can be applied at column i if and only if $\Delta h_i \geq G + g_1$ (see Fig. 1). Note that the rule application at i (which we also call *fire at i*) conserves the total number of sand grains. We also write $h \rightarrow h'$ when the fired column is not given, and denote by \rightarrow^* the reflexo-transitive closure of \rightarrow . When $\Delta h_i \geq G + g_1$, we say that column i is *unstable*. A configuration is *stable*, or a *fixed point*, if it has no unstable columns. Let \mathcal{C}_s be the set of stable configurations.

$$\mathcal{C}_s = \{h \in \mathcal{C} \mid \forall i \in \mathbb{N} : 0 \leq \Delta h_i < G + g_1\}.$$

We denote 0^ω the infinite sequence of 0. The models are non-deterministic: the rule is applied once at each time step. An example of successive applications for the rule (2, 1), ending in a fixed point, is given below:

$$h = (26, 0^\omega) \xrightarrow{0} (23, 2, 1, 0^\omega) \xrightarrow{0} (20, 4, 2, 0^\omega) \xrightarrow{0} (17, 6, 3, 0^\omega) \xrightarrow{0} (14, 8, 4, 0^\omega) \\ \xrightarrow{0} (11, 10, 5, 0^\omega) \xrightarrow{1} (11, 7, 7, 1, 0^\omega) \xrightarrow{2} (11, 7, 4, 3, 1, 0^\omega).$$

Since Δ is a bijection on $\mathbb{Z}_f^{\mathbb{N}}$, a configuration h can be encoded by Δh , or $\Delta^2 h$. The sequence Δh of slopes is a more local (spatially uniform) representation of configurations. For the fixed point of the example above, $\Delta h = (4, 3, 1, 2, 1, 0^\omega)$. When not specified, the sequence of slopes is the default manner to give a configuration throughout the paper.

Let $\mathcal{C}(h) = \{h' \mid h \rightarrow^* h'\}$ denote the set of configurations *reachable* from h . We focus on the case when $h = (N, 0^\omega)$ *i.e.* configurations with a finite number

$N \in \mathbb{N}$ of grains stacked on column 0, and no grains elsewhere. The set $\mathcal{C}((N, 0^\omega))$ is simply denoted by $\mathcal{C}(N)$.

The following theorem is obtained from *diamond property* plus *termination* (see for example [1] and [12]).

Theorem 1. $\mathcal{C}(h)$ endowed with \rightarrow has a graded lattice structure. Moreover, for any column i and any $h' \in \mathcal{C}(h)$, the number of firings of i during a sequence of transitions from h to h' does not depend of the chosen sequence of transitions.

In particular, even though decreasing sandpile models are non-deterministic, a unique fixed point, denoted by $\pi(h)$, is reached from any finite configuration h . The goal of this study is to understand the structure of $\pi(N) = \pi((N, 0^\omega))$. Configurations h' of $\mathcal{C}(h)$ admit another representation called *shot sequence*. It is the sequence $v = (v_i)_{i \in \mathbb{N}}$ where v_i is the number of times column i has been fired, from h to h' . The shot sequence of the stable configuration on our example is $v = (5, 1, 1, 0^\omega)$. Of course Δv or $\Delta^2 v$ can also encode a configuration of $\mathcal{C}(h)$. Surprisingly, the encoding $\Delta^2 v$ plays a fundamental role in the study of fixed points.

Remark 1. The classical one-dimensional sandpile rule corresponds to the 1-tuple (1), and the Kadanoff rule with parameter p to the p -tuple $(1, 1, \dots, 1)$. Important simplifications arise when the rule is a 1-tuple (g_1) , and a result similar to [9, 10] is obtained with the same technics (by grouping grains in little stacks of g_1 units). Consequently, in the present work, we focus on the case $p > 1$.

The Contribution. For a given decreasing sandpile rule \mathcal{R} , we are interested in the language (of sequences of slopes) of fixed points reached from initial configurations of the form $(N, 0^\omega)$,

$$\mathcal{L}_{\mathcal{R}} = \{\pi(N), N \in \mathbb{N}\} = \left(\bigcup_{N \in \mathbb{N}} \mathcal{C}(N) \right) \cap \mathcal{C}_s.$$

Experiments suggest that all words from this language present regular repetitions of short patterns. Interestingly, when $p > 1$ these regular repetitions do not cover the entire non null part of the fixed point, but emerge from the dynamics: all the grains are initially stacked on column 0, and their toppling towards a stable configuration lets the regularities appear only on the right of some relatively small but asymptotically infinite position. Figure 2 presents some computer simulations, which lead to the following conjecture.

Conjecture 1. For any rule $\mathcal{R} = (g_1, \dots, g_p)$, there exist words $w_l, w_c, w_{c'}, w_r, w_f$ and $w_{f'}$, each of length at most p , such that if $h = \pi(N)$ then there exists an integer n in $\mathcal{O}(\log(N))$ such that $(\Delta h_i)_{i \geq n} = w_l^* (w_c + w_{c'} + \epsilon) w_r^* (w_f + w_{f'}) 0^\omega$, where $*$ denotes finite repetitions, $+$ the *or*, and ϵ the empty word¹. There also exist words u_l, u_r on the alphabet $\{0, 1\}$, each of length at most p , such that if v

¹ $i \in \mathcal{O}(f(N))$ stands for $i \leq c f(N)$ for a suitable constant c .

is the shot sequence of $\pi(N)$, then there exists an integer n' in $\mathcal{O}(\log(N))$ such that $(\Delta^2 v_i)_{i \geq n'} = u_l^* (0 + \epsilon) u_r^* 0^\omega$.

Remark that the length $\mathcal{O}(\log(N))$ of the irregular left part is negligible compared to the length $\Theta(\sqrt{N})$ of the non null part of the sequences (given a decreasing rule, it is not hard to notice that this part is in $\Theta(\sqrt{N})$ because the fixed point is a rectangular triangle of area N and bounded slope, by stability from above and from below since there are no plateau of length greater than $p + 1$). Thus the conjecture gives a nearly exhaustive characterization of fixed points. Also remark that the first part of the conjecture is a (spectacular but direct) consequence of the second part. Thus, in the study, we will focus on the second part.

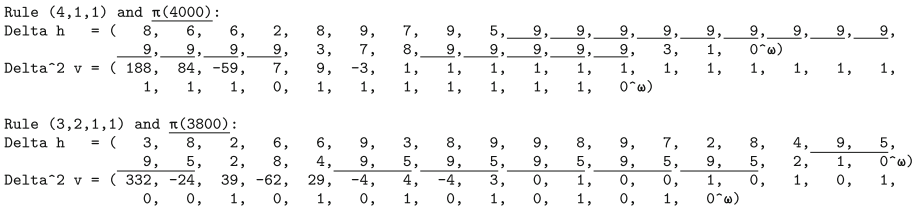


Fig. 2. Examples of fixed points with the regularly repeated pattern highlighted.

For the Kadanoff sandpile models, this conjecture has been proven in [15–17]. In the present paper we generalize in a large way the result about fixed points on Kadanoff sandpiles (note that Kadanoff sandpiles already generalize the classical one-dimensional model of [9,10]). We treat the extreme cases: when g_1 is sufficiently large to control the evolution (Theorem 4), and when g_1 is sufficiently small (Theorem 5).

First, the language $\mathcal{L}_{\mathcal{R}}$ is investigated via arguments of linear algebra linked to static constraints obtained from the definition of a fixed point (Sect. 2). By static constraints, we mean constraints telling that the configuration is stable, but nothing (or very few) about its creation process. This step gives an automaton whose set of accepting words contains $\mathcal{L}_{\mathcal{R}}$. Then, we use a dynamic and recursive construction of fixed points to refine the result. More precisely, the fixed point $\pi(N + 1)$ is obtained from $\pi(N)$ by adding a single grain on column 0 and performing all the possible firings until stability is reached (this process is called the $(N + 1)^{\text{th}}$ avalanche). Using combinatorial arguments, we formally prove that fixed points admit regular patterns, when the rule verifies arithmetic properties (Sect. 3).

Notice that these two steps informally correspond to the hybrid nature of sand. The algebraic part considers it as a (quasi) continuous fluid, while the combinatorial one considers it as a set of discrete atoms. To achieve this generalization, we made major improvements of the developments around the Kadanoff model. The first one is the use of stronger algebraic theorems about matrix norms, to ensure a convergence of the algebraic part. The second one is the introduction of the encoding $\Delta^2 v$ of configurations, which is very useful to describe

fixed points and avalanches, and allows to easily consider these two notions simultaneously. For the Kadanoff model, the automaton is quite simple (and so is the second step), thus the tools we introduce in the present work were not explicitly defined and used previously [15–17].

Finally, we discuss the results and conjecture their generalization to all decreasing sandpile models (Sect. 4). We strongly conjecture that our approach can succeed for any decreasing sandpile model. But, currently, we do not have a general proof, even if we are able to prove the conjectures for numerous particular rules.

2 Static Study of the Fixed Points

Equivalent Representations. Decreasing sandpile models can also be seen as *chip-firing games* (basically with a vertex for each column and a chip content corresponding to the slope) with a sink, what Dhar calls *abelian sandpile models* in [4]. The same notion of *equivalence class* applies. We define it on the set $\mathbb{Z}_f^{\mathbb{N}}$ of all ultimately null configurations (possibly increasing). For all $h, h' \in \mathbb{Z}_f^{\mathbb{N}}$, we have $h \equiv_{\mathcal{R}} h'$ if and only if h' can be reached from h by a sequence of *firings* and *anti-firings* (the reverse of a firing) on column indices in \mathbb{N} (configurations may have columns with negative sand content). It is reflexive, symmetric and transitive, hence an equivalence relation. The equivalence class of a configuration h is denoted $[h]$. Let

$$\Pi(h) = [h] \cap \mathcal{C}_s \quad \text{and} \quad \mathcal{L}'_{\mathcal{R}} = \bigcup_{N \in \mathbb{N}} \Pi(N).$$

As an example $(1, 1, 1, 0^\omega) \in \mathcal{L}'_{(1)} \setminus \mathcal{L}_{(1)}$. The developments of this section will give clues on the language $\mathcal{L}'_{\mathcal{R}}$. Note that $\pi(N) \in \Pi(N)$, hence $\mathcal{L}_{\mathcal{R}} \subseteq \mathcal{L}'_{\mathcal{R}}$, therefore the conclusions of Sect. 2 will apply to our main language of interest. The arguments are based on static equations about the final stability of the configurations of $\Pi(N)$. The main static relation we exploit is simple to notice. Let us canonically extend the definition of shot sequence to the configurations of $[N]$. This representation is obviously linked to the sequence of heights. The equality below simply expresses the grain balance on column i after firings and anti-firings, with respect to a decreasing rule $\mathcal{R} = (g_1, \dots, g_p)$ with $p > 1$.

$$\text{For all } i \geq p \text{ we have } h_i = -G v_i + g_1 v_{i-1} + g_2 v_{i-2} + \dots + g_p v_{i-p}. \quad (1)$$

In this section, we will basically make use of Eq. (1) and the constraint that the configurations in $\Pi(N)$ are stable, non-increasing, and ultimately null. From this we will first construct a recurrence equation describing the fixed point from left to right: given $(\Delta v_j)_{i \leq j < i+p}$ we will express $(\Delta v_j)_{i+1 \leq j < i+p+1}$. Then arguments of linear algebra will be employed in order to prove a convergence result on iterations of this system: columns on the right of a position in $\mathcal{O}(\log(N))$ have strong regularity properties. The results will eventually be expressed as an

automaton (depending on \mathcal{R}) recognizing the sequences of slopes of any configuration in $\Pi(N)$. Talking about the language $\mathcal{L}'_{\mathcal{R}}$ is not a goal in itself, but a way to express the fact that the arguments we exploit in this section are quite general: using static arguments we set up restrictions on the language of sequences of slopes for any configuration in $\Pi(N)$.

Section 3 will present a different and complementary kind of arguments, linked to the dynamics of the model, that will allow to refine the results to get a precise characterization of $\mathcal{L}_{\mathcal{R}}$ when the rule \mathcal{R} verifies some arithmetic properties.

The Perturbed Weighted Mean System. Equation (1) can be expressed in terms of Δh and Δv , leading to the *recurrence relation*

$$\Delta v_i = \frac{1}{G} (g_1 \Delta v_{i-1} + g_2 \Delta v_{i-2} + \dots + g_p \Delta v_{i-p}) - \frac{\Delta h_i}{G}. \tag{2}$$

In matricial form, it gives a system from \mathbb{Z}^p to \mathbb{Z}^p that we call *perturbed weighted mean system*:

$$\Delta V_i = M \Delta V_{i-1} - \frac{\Delta h_i}{G} K \tag{3}$$

with

$$\Delta V_i = \begin{pmatrix} \Delta v_{i-p+1} \\ \vdots \\ \Delta v_{i-1} \\ \Delta v_i \end{pmatrix} M = \begin{pmatrix} 0 & 1 & & 0 \\ & & \ddots & \\ 0 & 0 & & 1 \\ \frac{g_p}{G} & \frac{g_{p-1}}{G} & \dots & \frac{g_1}{G} \end{pmatrix} K = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

It is composed of two parts. First, a linear map $M : \mathbb{R}^p \rightarrow \mathbb{R}^p$ that

- Shifts all the values one row upward;
- For the last component, computes the mean of ΔV_{i-1} weighted by (g_1, \dots, g_p) .

Second, a discrete perturbation $\frac{\Delta h_i}{G}$ subtracted to the last component, so that $\Delta v_i \in \mathbb{Z}$. Intuitively, iterating the system, *i.e.*, computing the weighted mean and subtracting a bounded perturbation $((\Delta h_i)_{i \in \mathbb{N}} \in \mathcal{C}_s)$ will quickly tend to output values which are close to each other. In other words, the sequence $(\Delta V_i)_{i \geq p-1}$ will tend to uniform vectors. This is what we are about to prove. For convenience, let $m_i = \frac{1}{G}(g_p, \dots, g_1) \Delta V_i$ denote the weighted mean of ΔV_i , and \underline{m}_i (resp. \overline{m}_i) the minimal (resp. maximal) value of ΔV_i .

As a starting point of the system, we define $\Delta V_{-1} = {}^t(\frac{N}{g_p}, 0, \dots, 0, -v_0)$, emulating the fact that column 0 receives N units of sand grains. Now Eq. (2) and the perturbed weighted mean system hold for all $i \in \mathbb{N}$. For the configurations of $\Pi(N)$ we have

$$\frac{N}{G + g_1} \leq v_0 \leq \frac{N}{G}, \tag{4}$$

because from the initial configuration we have to perform at least the left bound number of firings (less and the configuration cannot be stable), and at most the

right bound number of firings (more and it is not possible to get a non-increasing and ultimately null configuration).

Regarding the discrete perturbation $\frac{\Delta h_i}{G}$, for all configurations of the set $\Pi(N)$ we have a relation for stability (left) and a relation for integrity (right).

$$0 \leq \frac{\Delta h_i}{G} < 1 + \frac{g_1}{G} \leq 2 \quad \text{and} \quad \Delta h_{i+1} \equiv G m_i \pmod{G}. \tag{5}$$

Note that consequently, for a given m_i there are at most two possible values of Δh_i that match these two constraints, and only one when $m_i \geq G + g_1$. This will be a key point in the construction of automata at the end of this section.

For the convergence of $(\Delta V_i)_{i \in \mathbb{N}}$ towards uniform vectors, we first prove that it converges in $\mathcal{O}(\log(N))$ iterations to vectors of bounded amplitude.

Lemma 1. *There exist a constant α and a $n_0 \in \mathcal{O}(\log(N))$ s.t. $\bar{m}_i - \underline{m}_i < \alpha$ for all $i > n_0$.*

Proof (sketch). Equation (4) implies that $\bar{m}_{-1} - \underline{m}_{-1}$ is in $\Theta(N)$. In order to prove that iterations of the perturbed weighted mean system tend to uniform vector, that is, each value will become closer and closer to the mean value, we take $M_i = {}^t(m_i, \dots, m_i) \in \mathbb{R}^p$ and study the sequence $(Z_i)_{i \in \mathbb{N}}$ where $Z_i = \Delta V_i - M_i$, which converges to $\mathcal{V}_p = {}^t(0, \dots, 0) \in \mathbb{Z}^p$. From Eq. (3) we get a relation of the form

$$Z_i = O Z_{i-1} - \frac{\Delta h_i}{G} L$$

where O is a contracting map: its spectral radius is strictly smaller than 1 (proved with a classical result due to Eneström and Kakeya, see for example [8]), and L has bounded norm. As a consequence, we can isolate the contracting map and the sum of perturbations,

$$Z_n = O^{n+1} Z_{-1} + \frac{1}{G} \sum_{i=0}^n \Delta h_i O^{n-i} L.$$

The left part of the sum tends exponentially to \mathcal{V}_p (see for example [13] for a discussion on contracting maps), and the right part is upper bounded by some constant $\alpha - 1$. Since the norm of Z_i is in the order of $\bar{m}_i - \underline{m}_i$, the norm of Z_{-1} is in $\Theta(N)$ and there exists an iteration n_0 in $\mathcal{O}(\log(N))$ such that the left term is strictly smaller than 1, leading to the result. \square

Secondly, it converges, at least linearly in the amplitude, to a sequence which is non-increasing and where two consecutive values are equal or differ by one.

Lemma 2. *There exists d in $\mathcal{O}(\bar{m}_i - \underline{m}_i)$, such that for all $k \geq i + d$, we have $\Delta v_k - \Delta v_{k+1} \in \{0, 1\}$. Moreover, $\Delta v_k - \Delta v_{k+1} = 1$ only if $m_i - \underline{m}_i < \frac{g_1}{G}$.*

Proof (sketch). The core argument of this proof is that, when $\overline{m}_i \neq \underline{m}_i$, the weighted mean m_i is strictly between \overline{m}_i and \underline{m}_i . The perturbation (bounded by Eq. (5)) subtracted to it then always leads to an integer Δv_{i+1} which is strictly below \overline{m}_i and greater or equal to $\underline{m}_i - 1$. As a consequence, the result holds if the sequence $(\Delta v_k)_{k \geq i+d}$ is non-increasing. In order to prove this, we can first notice that while Δv_{i+1} is above or equal to \underline{m}_i , it is still strictly below \overline{m}_i so ΔV_i tend linearly to uniform vectors. When it happens that $\Delta v_{i+1} = \underline{m}_i - 1$, the values embedded in ΔV_i must already be very close to each other, in order for the mean m_i to be just a little bit above \underline{m}_i , so that the perturbation (strictly smaller than 2 from Eq. (5)) subtracted to it can lead to a Δv_{i+1} strictly below \underline{m}_i . In this case, a careful look at the constraints on the closeness of the values embedded in ΔV_i allows to conclude. \square

The combination of Lemmas 1 and 2 gives the expected result, which we will express in term of the second derivative of the shot sequence. Let $\Delta^2 V_i = {}^t(\Delta^2 v_{i-p+1}, \dots, \Delta^2 v_{i-1})$. Note that $\Delta^2 V_i$ can be computed from ΔV_i , and so can the difference $m_i - \underline{m}_i$, with the function m defined as

$$m(q_1, \dots, q_{p-1}) = \sum_{1 \leq j < k \leq p} q_{p-j} g_k = \sum_{j=1}^{p-1} \left[q_{p-j} \left(\sum_{k=j+1}^p g_k \right) \right].$$

Property 1.

- For the product order, if $U < U'$ then $m(U) < m(U')$.
- If $\Delta v_i = \underline{m}_i$ then $m(\Delta^2 V_i) = G(m_i - \underline{m}_i)$.
- $m(0, 0, \dots, 0, 0, 1) = g_2 + g_3 + \dots + g_p$; $m(0, 0, \dots, 0, 1, 0) = g_3 + \dots + g_p$;
 $m(1, 0, \dots, 0, 0, 0) = g_p$; $m(1, 1, \dots, 1, 1, 1) = g_2 + 2g_3 + \dots + (p-1)g_p$.

Proposition 1. *There exists a column n_1 in $\mathcal{O}(\log(N))$, such that*

$$\text{for all } i \geq n_1 \text{ we have } \Delta^2 v_i \in \{0, 1\}.$$

Moreover, $\Delta^2 v_i = 1$ only if $m(\Delta^2 V_i) < g_1$, and

$$\text{for all } i \geq n_1 \text{ we have } \Delta h_{i+1} = m(\Delta^2 V_i) + \Delta^2 v_i G.$$

Proof. First part is a straight combination of Lemmas 1 and 2. For the middle part, when Δv is non-increasing we always have $\Delta v_i = \underline{m}_i$ and Property 1 applies. For the last part, from Relation (3) we have $\Delta h_{i+1} = G(m_i - \Delta v_{i+1})$. When $i \geq n_1$ the first part states that $\Delta^2 v_i$ equals 0 or 1 (recall that $\underline{m}_i = \Delta v_i$), and in both cases the conclusion is reached by applying Property 1. \square

Remark 1. To avoid confusion, remark that $\Delta^2 v_i$ is not the last component of $\Delta^2 V_i$, but the last component of $\Delta^2 V_{i+1}$.

The Automaton. Proposition 1 gives restrictions on the language $\mathcal{L}'_{\mathcal{R}}$. Let us now express these restrictions in the framework of automata theory. Given a decreasing sandpile rule, its *recurrence automaton* will be a Muller automaton (a kind of Büchi automaton with stronger accepting condition, see [14] for a definition) recognizing a subset of \mathcal{C}_s which includes $\mathcal{L}'_{\mathcal{R}}$ (asymptotically).

States of the automata correspond to vectors $\Delta^2 V_i \in \{0, 1\}^{p-1}$, and transitions to iterations of the perturbed weighted mean system. Depending on the value of $m(\Delta^2 V_i)$, Proposition 1 tells that there is one or two out-going transitions from a state. We label a transition with the value of Δh_{i+1} given by Proposition 1. We consider the whole set of states as potential initial states, and 0^{p-1} as the unique entry in the acceptance table of Muller automata, *i.e.*, runs must end in an infinite loop on the state 0^{p-1} , corresponding to the fact that all words of $\mathcal{L}'_{\mathcal{R}}$ are ultimately null (by definition).

Definition 2. Given a decreasing sandpile rule \mathcal{R} , let $\mathcal{A}_{\mathcal{R}}$ be its recurrence automaton, which is the Muller automaton whose set of states is $\{0, 1\}^{p-1}$, alphabet is \mathbb{N} , set of initial states is $\{0, 1\}^{p-1}$, acceptance table is $\{0^{p-1}\}$, and there exists a transition $(q_1, \dots, q_{p-1}) \xrightarrow{a}_{\mathcal{R}} (q'_1, \dots, q'_{p-1})$ if and only if

- $(q'_1, \dots, q'_{p-2}) = (q_2, \dots, q_{p-1})$;
- either $q'_{p-1} = 0$
or $q'_{p-1} = 1$ and $m(q) < g_1$;
- $a = m(q) + q'_{p-1}G$.

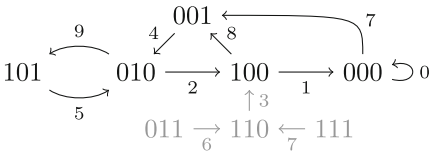


Fig. 3. Recurrence automata $\mathcal{A}_{\mathcal{R}}$ for $\mathcal{R} = (3, 2, 1, 1)$, where irrelevant states are shaded. Note that it is in accordance with the example from Fig. 2.

An example of recurrence automaton is given on Fig. 3. Theorem 2 rephrases Proposition 1 in order to characterize $\mathcal{L}'_{\mathcal{R}}$.

Theorem 2. Let $\mathcal{L}(\mathcal{A}_{\mathcal{R}})$ denotes the language of infinite words recognized by $\mathcal{A}_{\mathcal{R}}$. For all $h \in \Pi(N)$, there exists a column n_2 in $\mathcal{O}(\log(N))$ such that

$$(\Delta h_i)_{i \geq n_2} \in \mathcal{L}(\mathcal{A}_{\mathcal{R}}).$$

Theorem 2 is equally valid if we simplify recurrence automata to their *ultimately relevant* states, *i.e.*, those belonging to a directed cycle (note that 0^{p-1} is always an ultimately relevant state with a loop labelled with slope value 0). Let $\mathcal{A}'_{\mathcal{R}}$ denote these simplified automata. Languages $\mathcal{L}(\mathcal{A}'_{\mathcal{R}})$ contain suffixes of the languages we are interested in. It is not tight enough to fit $\mathcal{L}_{\mathcal{R}}$, not even exactly $\mathcal{L}'_{\mathcal{R}}$, but can nevertheless be considered as an important progress compared to \mathcal{C}_s . Furthermore, the developments so far are only based on *static* relations coming from the model definition. Finally, let us recall that $w(\pi(N)) \in \Theta(\sqrt{N})$ for all rule and number of grains N , therefore $\mathcal{L}(\mathcal{A}'_{\mathcal{R}})$ contains prefixes of $\mathcal{L}'_{\mathcal{R}}$ that asymptotically account for the whole fixed points. Note that this is not an equality: $\mathcal{A}'_{\mathcal{R}}$ recognizes other words.

3 Dynamic Study of the Fixed Points

Avalanches. This section complements the developments around $\mathcal{L}_{\mathcal{R}}$ presented so far ($p > 1$), with arguments linked to the dynamics of decreasing sandpile models. It is based on the fact that for a given rule, $\pi(0), \pi(1), \pi(2), \dots, \pi(N)$ must all belong to $\mathcal{L}(\mathcal{A}'_{\mathcal{R}})$ (starting from some index n_2 in $\mathcal{O}(\log(N))$). Let us present the notion of *avalanche*, and two cases where combinatorial arguments allow to characterize asymptotically $(\Delta^2 v_i)_{i \in \mathbb{N}}$ (hence $(\Delta h_i)_{i \in \mathbb{N}}$ when g_1 is above some upper threshold or below some lower threshold (note that the second case includes Kadanoff sandpile models).

Let $c^{\downarrow 0}$ denote the configuration obtained from c by adding one grain on column 0. If $c \xrightarrow{i_1} \dots \xrightarrow{i_k} c'$ then $c^{\downarrow 0} \xrightarrow{i_1} \dots \xrightarrow{i_k} c'^{\downarrow 0}$. This is in particular true for $c = (N, 0^\omega)$ and $c' = \pi(N)$, thus fixed points can inductively be computed with

$$\pi(N + 1) = \pi(\pi(N)^{\downarrow 0}),$$

from $\pi(0) = (0^\omega)$, repeating N times the addition of one grain followed by the stabilization process. We will now use this point of view. Let the $(N + 1)^{th}$ *avalanche* be the sequence $(a_i)_{i \geq 0}$ where a_i denotes the number of times that column i has been fired in the stabilization process from $\pi(N)^{\downarrow 0}$ to $\pi(N + 1)$. Let plain h, v (resp. primed h', v') denote representations of $\pi(N)$ (resp. $\pi(N + 1)$),

$$(v'_i - v_i)_{i \geq 0} = (a_i)_{i \geq 0} \quad \text{and} \quad (\Delta^2 v'_i - \Delta^2 v_i)_{i \geq 0} = (\Delta^2 a_i)_{i \geq 0}. \tag{6}$$

Avalanches are studied in [15], which is generalized as follows.

Theorem 3.

1. For each integer $i \geq 1$, we have $a_i \in \{0, 1\}$.
2. If $a_j = a_{j+1} = \dots = a_{j+p-1} = 0$ for some $j \in \mathbb{N}$, then $a_k = 0$ for all $k \geq j$.
3. If $\Delta h_{j+r} + g_{r+1} < G + g_1$ for some $j \in \mathbb{N}$ and all $0 \leq r \leq p - 1$, then integer j satisfies condition 2 above.

Proof (sketch). Arguments similar to [15], telescoping sum for third point. \square

First, note that Proposition 1 and Eq. (6) imply that there exists $n_2 \in \mathcal{O}(\log(N))$ such that $(\Delta^2 a_i)_{i \geq n_2} \in \{-1, 0, 1\}^\omega$, hence there is no factor 010 or 101 in $(a_i)_{i \geq n_2}$. A simple case by case study of what does the value of $\Delta^2 a_i$ implies on a_i, a_{i+1} and a_{i+2} shows that, up to a one unit shift, the patterns 1(-1) and (-1)1 of $(\Delta^2 a_i)_{i \geq n_2}$ are delimiters of the intervals of 0 and 1 in $(a_i)_{i \geq n_2}$. Moreover, the length of intervals of 0 in $(a_i)_{i \geq n_2}$ is at most $p - 1$ (except the ultimate 0^ω). Basic considerations of this type lead to the following proposition (* is the Kleene star denoting finite repetitions).

Proposition 2. $(\Delta^2 a_i)_{i \geq n_2}$ is suffix of a sequence in $(0^* 1(-1) 0^* (-1) 1)^* 0^\omega$.

We will have two main arguments: the stopping condition of Theorem 3, and the compatibility between $\pi(N)$ and $\pi(N + 1)$ (Proposition 1 and Theorem 2). For $1 \leq i \leq p - 1$, let $E_i \in \{0, 1\}^{p-1}$ have all null components except the i^{th} .

Case $g_1 > M(1, 1, \dots, 1)$. In this case two transitions are possible from any state in the automaton *i.e.*, Theorem 2 gives no constraint other than $\Delta^2 v_i \in \{0, 1\}$ for all $i \geq n_2$. Nevertheless we show that, on the right of column n_2 , the avalanche process fires a set of consecutive columns and stops.

Lemma 3. *If $\exists k \in \mathbb{N}$ such that $\Delta^2 v_{n_2+k} = 0$, then $a_j = 0$ for $j \geq n_2 + k + 1$.*

Proof (sketch). The goal is to reach the stopping condition of Theorem 3. From the hypothesis, $\Delta h_{n_2+k+1} = m(\Delta^2 V_{n_2+k}) + \Delta^2 v_{n_2+k} G \leq m(1, \dots, 1) < g_1$. For $2 \leq r \leq p$ we have $\Delta^2 V_{n_2+k+r-1} \leq {}^t(1, \dots, 1) - E_{p+1-r}$ and $m(E_{p+1-r}) = g_r + g_{r+1} + \dots + g_p$. Since m is a linear map, $\Delta h_{n_2+k+r} + g_r \leq m(1, \dots, 1) + g_1 + g_2 + \dots + g_r < g_1 + G$ and from Theorem 3 the avalanche stops (item 3). \square

Theorem 4. $(a_i)_{i \geq n_2} \in 1^* 0^\omega$ and $(\Delta^2 v_i)_{i \geq n_2} \in 1^* (0 + \epsilon) 1^* 0^\omega$.

Proof (sketch). The result is proven by induction on N , the base case is obvious. For the induction, we have $(\Delta^2 v_i)_{i \geq n_2} \in 1^k (0 + \epsilon) 1^{k'} 0^\omega$. From Proposition 2 and Eq. 6, we have either $(\Delta^2 a_i)_{i \geq n_2} = 0^\omega$, or $(\Delta^2 a_i)_{i \geq n_2} = 0^{k-1} (-1) 1 0^\omega$, otherwise it contradicts Proposition 1. In both cases the statement holds. \square

Case $m(E_r) \leq G_1 < M(E_{r+1})$ with $r + 1 \leq \frac{p}{2}$. Note that it is always true that $m(E_1) = g_p \leq g_1$. In this case, we will prove two preliminary lemmas and a description of $(\Delta^2 v_i)_{i \geq n_2}$ similar to Theorem 4.

Lemma 4. *In $(\Delta^2 v_i)_{i \geq n_2}$ two 1 are separated by at least $p - r - 1$ values 0.*

Proof (sketch). In terms of the automaton $\mathcal{A}_{\mathcal{R}}$, while the state q ends with strictly less than $p - r - 1$ values 0, there is only one transition (adding a new 0 at the end of q) because $m(q) \geq g_1$ (the minimal case for m is $q = 0^r 1 0^{p-r-2} = E_{r+1}$). The result follows by induction. \square

The next Lemma tells that $p - r$ values 0 in $\Delta^2 v$ stop the avalanche.

Lemma 5. *If $\Delta^2 v_j = 1$ and $\Delta^2 v_{j+p-r} = 0$ for $j > n_2 + p$, then $(a_i)_{i \geq j+2} = 0^\omega$.*

Proof (sketch). The goal is again to reach the stopping condition of Theorem 3. From Lemma 4, the hypothesis, and because $m(E_{p-1}) + g_1 = G$, we have $\Delta h_{j+2} + g_1 = m(\Delta^2 V_{j+1}) + g_1 \leq m(E_{p-1}) + m(E_{r-1}) + g_1 < G + g_1$. It follows from similar arguments that $\Delta h_{j+k} + g_{k-1} < G + g_{k-1}$ for $2 \leq k \leq p + 1$, thus Theorem 3 (item 3) applies. \square

Theorem 5. $(a_i)_{i \geq n_2+p} \in 1^* 0^\omega$ and $(\Delta^2 v_i)_{i > n_2+p}$ is suffix of an element in $(0^{p-r-1} 1)^* (0 + \epsilon) (0^{p-r-1} 1)^* 0^\omega$.

Proof (sketch). The result is proven by induction on N . The base case is obvious, and the induction is obtained from Eq. (6), and the constraints given by Proposition 1, Lemmas 4 and 5. \square

4 Conclusions and Perspectives

This paper has explored the language $\mathcal{L}_{\mathcal{R}}$ of the sequences of slopes of fixed points in decreasing sandpile models described by a transition rule \mathcal{R} . After a general development based on linear algebra, in two cases, Theorems 4 and 5 give precise asymptotic characterizations of $\Delta^2 v$, which apply to the sequences of slopes (via Proposition 1). We conjecture that a similar characterization holds for any decreasing sandpile rule (see Conjecture 1).

Note that for the two cases we solved, we have $w_l = w_r$ (from the labeling of recurrence automata and Theorem 2), but this is not always the case. For example, we strongly conjecture that for the rule $(6, 1, 1, 1, 1)$, the sequence $\Delta^2 v$ is asymptotically of the form $(0011)^{k_l} (0 + \epsilon)(01)^{k_r}$ and a subsequent avalanche leads to $(1100)^{k_l-1} 0(01)^{k_r+2}$ when the central part was 0, and to $(1100)^{k_l+1} (01)^{k_r-2}$ when the central part was ϵ . Hence w_l and w_r may differ.

The proof technique works in several stages, that may highlight the fact that sandpiles are at the edge between discrete and continuous systems. Lemma 1 uses arguments of linear algebra corresponding to a rough continuous nature, while Lemma 2 and Sect. 3 refine the study with precise combinatorial arguments corresponding to discrete dynamical phenomena.

The structure of avalanches is also an interesting point of view. We propose the following conjecture, satisfied for the cases treated completely in Sect. 3.

Conjecture 2. For any decreasing sandpile rule \mathcal{R} and any positive integer N , there exists n in $\mathcal{O}(\log(N))$ such that the N^{th} avalanche verifies $(a_i)_{i \geq n} \in 1^* 0^\omega$.


According to the conjectures above, fixed points and avalanches are characterized on the right of some column n in $\mathcal{O}(\log(N))$ compared to their width in $\Theta(\sqrt{N})$ *i.e.*, asymptotically completely. Though its relative size tends to be null, the unknown part between 0 and n is not bounded. The conjectures mean that as we add grains one by one, they trigger avalanches that let grains create and maintain regular patterns, after a transitional phase of unbounded length. Let us finish on a question: should this process be called *self-organized emergence*?

References

1. Baader, F., Nipkow, T.: Term Rewriting and all that. University Press, Cambridge (1998)
2. Bak, P., Tang, C., Wiesenfeld, K.: Self-organized criticality: an explanation of the $1/f$ noise. Phys. Rev. Letter **59**, 381–384 (1987)
3. Chazelle, B.: Natural algorithms. In: SODA, pp. 422–431 (2009)
4. Dhar, D.: Theoretical studies of self-organized criticality. Phys. Stat. Theor. Phys. **369**(1), 29–70 (2006)
5. Durand-Lose, J.O.: Parallel transient time of one-dimensional sand pile. Theor. Comput. Sci. **205**(1–2), 183–193 (1998)
6. Formenti, E., Masson, B.: On computing fixed points for generalized sand piles. Int. J. Unconventional Comput. **2**(1), 13–25 (2005)

7. Formenti, E., Van Pham, T., Phan, H.D., Tran, T.H.: Fixed point forms of the parallel symmetric sandpile model. *Theor. Comput. Sci.* **533**, 1–14 (2014)
8. Gardner, R.B., Govil, N.K.: Some generalizations of the eneström-akeya theorem. *Acta Math. Hung.* **74**(1–2), 125–134 (1997)
9. Goles, E., Kiwi, M.: One-dimensional sandpiles, cellular automata and related models, pp. 169–185. *Nonlinear Phenomena in Fluids, Solids and Other Complex Systems* (1991)
10. Goles, E., Kiwi, M.: Games on line graphs and sand piles. *Theor. Comput. Sci.* **115**(2), 321–349 (1993)
11. Goles, E., Latapy, M., Magnien, C., Morvan, M., Phan, H.D.: Sandpile models and lattices: a comprehensive survey. *Theor. Comput. Sci.* **322**(2), 383–407 (2004)
12. Goles, E., Morvan, M., Phan, H.D.: The structure of a linear chip firing game and related models. *Theor. Comput. Sci.* **270**(1–2), 827–841 (2002)
13. Katok, A., Hasselblatt, B.: *Introduction to the Modern Theory of Dynamical Systems*. University Press, Cambridge (1996)
14. Muller, D.E.: Infinite sequences and finite machines. In: *SWCT*, pp. 3–16 (1963)
15. Perrot, K., Rémila, E.: Kadanoff sand pile model. Avalanche structure and wave shape. *Theor. Comput. Sci.* **504**, 52–72 (2013)
16. Perrot, K., Rémila, É.: Emergence of wave patterns on kadanoff sandpiles. In: Pardo, A., Viola, A. (eds.) *LATIN 2014*. LNCS, vol. 8392, pp. 634–647. Springer, Heidelberg (2014)
17. Perrot, K., Rémila, E.: Strong emergence of wave patterns on kadanoff sandpiles. submitted to a journal (2015)

Lost in Self-Stabilization

Damien Regnault¹ and Éric Rémila²

¹ IBISC, EA4526, Université d'Évry Val-d'Essonne,
91037 Évry, France

damien.regnault@ibisc.univ-evry.fr

² GATE LSE (UMR CNRS 5824), Université de Lyon,
Site Stéphanois, 42023 Saint-Etienne, France
eric.remila@univ-st-etienne.fr

Abstract. Let t_a and t_b a pair of relatively prime positive integers. We work on chains of $n(t_a + t_b)$ agents, each of them forming an upper and rightward directed path of the grid \mathbb{Z}^2 , from $O = (0, 0)$ to $M = (nt_a, nt_b)$. We are interested on evolution rules such that, at each time step, an agent is randomly chosen on the chain and is allowed to jump to another site of the grid, with preservation of the connectivity of the chain, and the endpoints. The rules must be local, i.e. the decision of jumping or not only depends on the neighborhood of fixed size s of the randomly chosen agent, and not on the parameters t_a, t_b, n .

In the paper, we design such a rule which, starting from any chain which does not crosses the continuous line segment $[O, M]$, reorganizes the chain by iterate applications of the rule, in such a way such that it stabilizes into one of the best possible approximations of $[O, M]$. The stabilization is reached after $O(n(t_a + t_b))^4$ iterations.

1 Introduction

1.1 The Result

In this paper, we define and analyze a random process whose interest is at the crossroad of many different domains. Among all the different interpretations of our work, we choose a simple graphical representation to ease the reading of the article. Our model is precisely described in Sect. 2, but we now give an informal presentation. We work on a 2D discrete grid of size $A \times B$. We are given a chain of $A + B$ agents, which forms an upper and rightwards directed path between the opposite endpoints of the grid, of coordinates $(0, 0)$ and (A, B) such that two successive agents are neighbors. We want to design a rule such that, at each time step, an agent is randomly chosen and is allowed to jump in an other site of the grid, with preservation of the connectivity of the chain. The goal of the process is to reorganize the chain in such a way such that it stabilizes into the best possible approximation of the continuous line, of slope $\frac{B}{A}$, passing

This work is partially supported by Programs ANR Dynamite, Quasicool (ANR-12-JS02-011-01) and IXXI (Complex System Institute, Lyon).

by the opposite endpoints, *i.e.* the path which is included in a strip of slope $\frac{B}{A}$ with the lowest possible thickness.

We present here a distributed algorithm which achieves this goal with very few requirements. All modifications are decided locally by the agents which are memoryless, disoriented and have a limited range of communication. The decisions only rely on the relative position of the closest neighboring agents. The resulting dynamics stabilizes into the desired position in $O((A+B)^4)$ time steps in expectation up to two constraints.

The first constraint concerns the local sight s of each agent (*i.e.* an agent can only observe sites which are at distance at most s from its own position). Let (t_a, t_b) be the pair of relatively prime positive integers such that $\frac{t_b}{t_a} = \frac{B}{A}$. We prove that our process succeeds as soon as $t_a + t_b \leq s$. We also show that this bound on the sight is almost tight: a sight of at least $t_a + t_b - 1$ is needed to self-stabilize a discrete line of slope $\frac{B}{A}$.

The second constraint is that initially all agents are in the same side of the continuous line linking the two endpoints. We conjecture that this constraint can be removed by allowing a larger sight to sites close to endpoints. Nevertheless, in the general case, we can ensure a good approximation of the continuous line. Moreover, if we avoid boundary effects by identifying the endpoints of the chain, (creating a cycle) we also get an optimal approximation of the continuous line.

1.2 Contexts

The algorithm developed here draw inspiration from many different works. Due to space constraints, we cannot develop in details all these interconnections. The model was initially developed to study the *cooling process* of crystals. At high temperature, the arrangement of atoms is chaotic but when the temperature decreases atoms self-stabilize into an ordered structure: a crystal. Crystals are commonly modeled by tilings [8]. In a set of studies [2,3] we developed a model which transforms an unordered tiling into an ordered one. This article extends a previous study [7] to deal with non-identical initial proportions of atoms. A source of inspiration to develop the solution proposed here came from the GO-TO-THE-MIDDLE algorithm to solve the robot chain problem [4] in distributed computing. For this community, our work can be seen as a discretization of GO-TO-THE-MIDDLE. Note that this discretization of a continuous algorithm is far from trivial. As a proof of this complexity, we show that if the space is discrete then the robots need to know the position of more than two of their nearest neighbors to be efficient (see Theorem 4) unlike the continuous case where this condition is not necessary. Also, it is important to note that this problem is heavily linked to language theory. Indeed a discrete line on the plane can be represented as a word. Words representing a discrete approximation of a continuous line of rational slope are called Christoffel word [1] (Sturm words are a discrete approximation of continuous lines of irrational slope). Also, for dealing with technical details during the analysis of our algorithm, we use a *height function* [9,11] on tilings and some previous studies on probabilistic cellular automata [6].

In Sect. 2, we define formally our problem. In Sect. 3, we present our process and its main properties. We show that the obtained dynamics quickly converges to a solution which is close to the objective line, and give our results in Sect. 4¹. Finally in Sect. 5, we present some questions left open by this paper.

2 The Model

2.1 Configurations and Associated Words

Let (A, B) be a pair of positive integers. We state $\gcd(A, B) = n$, $t_a = \frac{A}{n}$, $t_b = \frac{B}{n}$, $t_{a,b} = t_a + t_b$ and $m = A + B = n t_{a,b}$. The ratio $\frac{B}{A} = \frac{t_b}{t_a}$ represents the slope of the continuous line of equation: $-t_b x + t_a y = 0$, passing by $(0, 0)$ and (A, B) , that we wish to approximate. This line is called the *ideal line*. In this paper, elements of \mathbb{N}^2 are called *sites*.

We state: $a = (1, 0)$, $b = (0, 1)$ and $\Sigma = \{a, b\}$. A *configuration* c is a sequence (c_0, \dots, c_m) of sites such that $c_0 = (0, 0)$, $c_m = (A, B)$ and for each $i < m$, either $c_{i+1} - c_i = a$ or $c_{i+1} - c_i = b$. The set of configurations is denoted by \mathbb{P} . The word w associated to the configuration c is the word $w = w_1 w_2 \dots w_m$ of Σ^m , such that; for each i of $\{1, 2, \dots, m\}$, w_i is the value of $c_i - c_{i-1}$.

For each word w of Σ^* , let $|w|_a$ (respectively $|w|_b$) denote the number of letters a (respectively b) in w , and $|w| = |w|_a + |w|_b$. If w is associated to a configuration c , then we have $|w|_a = A$; $|w|_b = B$. Conversely, for each word w of Σ^m , with $|w|_a = A$ and $|w|_b = B$, can be associated to a unique configuration c of \mathbb{P} , *i.e.* \mathbb{P} and $\{w \in \Sigma^m : |w|_a = A \text{ and } |w|_b = B\}$ are in bijection.

For each site $c = (x, y)$, we define the *height* $h(c) = -t_b x + t_a y$. In the following, we will extensively use the following properties:

Property 1. Let c and c' denote two sites.

- $h(c + a) = h(c) - t_b$ and $h(c + b) = h(c) + t_a$,
- we have $h(c) = h(c')$ if and only if there exists an integer k such that $c' - c = k(t_a, t_b)$,
- we have $h(c) \equiv h(c') \pmod{t_{a,b}}$ if and only if there exists two integers k and k' such that $c' - c = k(t_a, t_b) + k'(-a + b)$.

In particular, for each i of $\{0, \dots, m\}$, the value $h(c_i) \pmod{t_{a,b}}$ does not depend on the configuration c but only on $i \pmod{t_{a,b}}$.

For a configuration c , we define $h_{min}(c) = \min\{h(c_i), 0 \leq i \leq m\}$ and $h_{max}(c) = \max\{h(c_i), 0 \leq i \leq m\}$. Thus the configuration c is included the closed strip limited by lines (of slope $\frac{t_b}{t_a}$) of equations $-t_b x + t_a y = h_{min}(c)$ and $-t_b x + t_a y = h_{max}(c)$. Moreover this strip is the smallest one among all strips limited by lines of slope $\frac{t_b}{t_a}$. The *thickness* $\Delta_h(c)$ of the configuration c is defined by $\Delta_h(c) = h_{max}(c) - h_{min}(c)$. The properties of values modulo $t_{a,b}$ imply that $\Delta_h(c) \geq t_a + t_b - 1$.

¹ An extended version of the analysis of Sect. 4 can be found in <http://arxiv.org/abs/1410.7669>.

Among all configurations of \mathbb{P} , we will focus on the ones which are good approximations of the ideal line, *i.e.* configurations c such that $\Delta_h(c)$ is minimal. The words associated to these configurations are called *Christoffel words* and are extensively studied [1]. We use the following definition of Christoffel words, which is the most practical in our context: the word w associated to a configuration c is a Christoffel word of slope $\frac{t_b}{t_a}$ if and only if $\Delta_h(c) = t_a + t_b - 1$. We will also say that, in such a case, c is a *Christoffel configuration*.

2.2 Local Transition Rules

Configurations are static objects, now we introduce a way to modify them locally. Consider a configuration c and $2 \leq i \leq m$, the configuration c' obtained by *flipping* letters $i - 1$ and i in c is defined as follows: consider the words w, w' of Σ^m where w is the word associated to configuration c and w' is obtained by flipping letters w_{i-1} and w_i in w , *i.e.* $w'_{i-1} = w_i, w'_i = w_{i-1}$ and for all $j \in \{1, \dots, m\} \setminus \{i - 1, i\}$, $w'_j = w_j$; then c' is the configuration associated to w' . Note that for all $0 \leq j \leq m$ such that $j \neq i$ we have $c_j = c'_j$ and:

- if $w_{i-1} = a$ and $w_i = b$ then $c'_j = c_j - a + b$ and the flip is called *increasing*,
- if $w_{i-1} = b$ and $w_i = a$ then $c'_j = c_j + a - b$ and the flip is called *decreasing*.

We will also denote this operation as “flipping in site c_i ” for a configuration c . Consider a configuration c , doing an increasing (resp. decreasing) flip in c_i increases (resp. decreases) the height of this site from $t_{a,b}$ units.

We fix a positive integer s . Let $\Sigma^{\leq s}$ denote the set of non-empty words on Σ of length at most s , *i.e.* $\Sigma^{\leq s} = \bigcup_{s'=0}^s \Sigma^{s'}$. Let c be a configuration, w its associated word, and i such that $1 \leq i \leq m - 1$. The *right word* $w^r(c, i)$ in i for c is the word $w(c, i)$ of $\Sigma^{\leq s}$ defined by $w^r(c, i) = w_{i+1} w_{i+2} \dots w_{i+s}$ when $i + s \leq m$, and $w^r(c, i) = w_{i+1} w_{i+2} \dots w_m$ when $i + s > m$. In a similar way, the *left word* $w^l(c, i)$ in i for c is the word $w^l(c, i)$ of $\Sigma^{\leq s}$ defined by $w^l(c, i) = w_i w_{i-1} \dots w_{i-s+1}$ when $i \geq s$, and $w^l(c, i) = w_i w_{i-1} \dots w_1$ when $i < s$. We choose to write $w^l(c, i)$ reversing indices, since we adopt the point of view of a processor located in c_i , which reads words starting from its own position.

A *local transition rule* of sight s is given by a function $\delta : (\Sigma^{\leq s})^2 \rightarrow \{0, 1\}$. Given a configuration c , we say that the site c_i is *active* in c when $\delta(w^l(c, i), w^r(c, i)) = 1$. Otherwise, the site c_i is *inactive*.

Notice that from our formalism, the activity status of a site c_i does not depend on any global parameters t_a, t_b, n, m , the integer i and the position of c_i on the grid. We say that the rule is *local*, *anonymous* and that c_i is *partially lost*. We say “partially lost” since there exist rules which allow c_i to use some local elements of orientation: the site does not know its own position, but, nevertheless, it can possibly make difference between the top and the bottom, and between clockwise and counterclockwise senses, and use these informations to choose its activity status. We want to use a rule for which sites are “completely lost”, in the sense that the rule does not use the informations above. Formally, a rule δ

is *totally symmetric* when for each pair (w, w') of $(\Sigma^{\leq s})^2$, we have $\delta(w, w') = \delta(w', w) = \delta(g(w), g(w'))$, where g is the word morphism on Σ^* such that $g(a) = b$ and $g(b) = a$.

By abuse of notation, for a configuration c (which may be a random configuration), we design by $\delta(c)$ the random configuration obtained as follows: a number i of $\{1, 2, \dots, m - 1\}$ is selected uniformly at random, and if the corresponding site c_i is active, then it is flipped, otherwise $\delta(c) = c$.

The transition rule δ introduces a discrete Markovian process on configurations: let c^t design the configuration at time t , c^0 is the *initial configuration*. The configuration at time $t + 1$ is a random variable defined by $c^{t+1} = \delta(c^t)$.

3 The Specific Transition Rule

Our aim is to specify the rule δ in order to construct a *coalescence process*, *i.e.* a totally symmetric local transition rule such that:

- any initial configuration will reach a Christoffel configuration of slope $\frac{t_b}{t_a}$ in polynomial expected time,
- all Christoffel configurations of slope $\frac{t_b}{t_a}$ are *stable*, *i.e.* have no active site.

We will now describe our transition rule δ . We use an auxiliary rule δ' such that, for each pair (w, w') of $(\Sigma^{\leq s})^2$, $\delta'(w, w') = \delta'(g(w), g(w'))$. We define δ by: for each pair (w, w') of $(\Sigma^{\leq s})^2$, we have $\delta(w, w') = \max\{\delta'(w, w'), \delta'(w', w)\}$. This ensures δ to be totally symmetric.

To completely define δ' , since for each pair (w, w') of $(\Sigma^{\leq s})^2$, $\delta'(w, w') = \delta'(g(w), g(w'))$, it suffices to define $\delta'(w, w')$ when $w'_1 = b$. We have several constraints to have $\delta'(w, w') = 1$. If those constraints are simultaneously satisfied then, $\delta'(w, w') = 1$. Otherwise $\delta'(w, w') = 0$. These constraints are stated and explained below.

Sight Constraint: the first constraint is that:

$$|w'| = s. \tag{1}$$

The interpretation is clear, c_i must have a visibility at least s on its right side.

Weak Thickness Constraint: This second constraint is the heart of the process. It ensures that the thickness of the configuration is not increasing wherever the flip is done. This is not trivial without the knowledge of (t_a, t_b) .

For $1 \leq i \leq s$, we define a'_i (respectively b'_i) as the number of a (respectively b) in the prefix of length i of w' , *i.e.* the word $w'_1 w'_2 \dots w'_i$; and for $1 \leq j \leq |w|$, we define a_j (respectively b_j) as the number of a (respectively b) in the prefix of length j of w . *i.e.* the word $w_1 w_2 \dots w_j$. We define (r_a, r_b) as (a'_i, b'_i) , with $\frac{b'_i}{a'_i}$ minimum (with the convention $\frac{b}{0} = +\infty$). In case of tie, we take the pair with the lowest index i (according to the previous convention, for $w' = bb \dots b$, we take $(r_a, r_b) = (1, 0)$).

The second constraint necessary to possibly have $\delta'(w, w') = 1$ is:

$$\exists j \in \{1, 2, \dots, s\} \mid r_b a_j - r_a b_j \geq r_a + r_b. \tag{2}$$

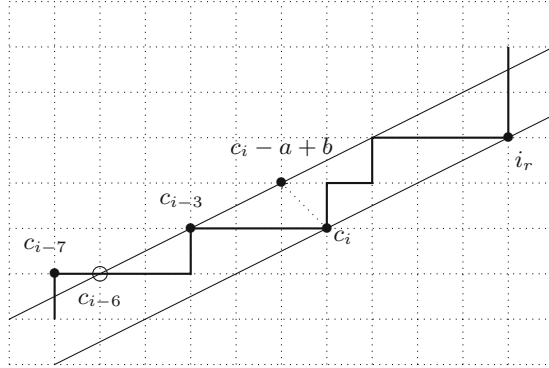


Fig. 1. The site c_i has a sight of 8 and this scheme represents the test to determine if c_i is active or not. Sites c_{i-3} , c_{i-6} and c_{i-7} satisfy the weak thickness constraint ensuring that the thickness will not increase by doing a flip in c_i . Nevertheless, site c_{i-6} does not satisfy the strong thickness constraint (introduced later) needed to ensure a polynomial convergence time on expectation.

Notice that $-r_b x + r_a y \geq r_a + r_b + h(c_i)$ is an equation of the half-plane limited by the line ℓ'_r of slope $\frac{r_b}{r_a}$ passing by $c_i - a + b$, and not containing c_i . The condition claims that there exists a site c_{i-j} , with $1 \leq j \leq s$, which is element of this half-plane, *i. e.* is over the limit line (Fig. 1).

Definition 1. Let s be a positive integer. A pair (u, v) of \mathbb{N}^2 is visible by s if $u + v \leq s$.

Lemma 1. Assume that (t_a, t_b) is visible by s . Let c be a configuration and $i \in \{1, 2, \dots, m - 1\}$ such that, if we state $(w^l(c, i), (w^r(c, i))) = (w, w')$, then (w, w') satisfies the two constraints above, and let j be an integer allowing to satisfy the thickness constraint.

- If $\frac{r_b}{r_a} > \frac{t_b}{t_a}$, then $h(c_i) + t_a + t_b \leq h(c_{i+t_a+t_b})$,
- If $\frac{r_b}{r_a} < \frac{t_b}{t_a}$, then $h(c_i) + t_a + t_b < h(c_{i-j})$,
- If $\frac{r_b}{r_a} = \frac{t_b}{t_a}$, then $h(c_i) + t_a + t_b \leq h(c_{i-j})$. Moreover, in this case we have the equivalence:

$$r_b a_j - r_a b_j = r_a + r_b \iff h(c_i) + t_a + t_b = h(c_{i-j}).$$

Proof. The two first cases are illustrated in Fig. 2. If $\frac{r_b}{r_a} > \frac{t_b}{t_a}$, then, first, since $i \equiv (i+t_a+t_b) \pmod{t_a+t_b}$, we have $h(c_i) \equiv h(c_{i+t_a+t_b}) \pmod{t_a+t_b}$. Thus, it suffices to prove that $h(c_{i+t_a+t_b}) > h(c_i)$. We have $h(c_{i+t_a,b}) = h(c_i) - a'_i t_b + b'_i t_a$.

Thus, if $a'_t = 0$ (which implies $b'_t = t_{a,b}$) then we are done. Otherwise, we have $\frac{b'_t}{a'_t} \geq \frac{r_b}{r_a} > \frac{t_b}{t_a}$, which gives

$$\begin{aligned} h(c_{i+t_{a,b}}) &= h(c_i) - a'_t t_b + b'_t t_a = h(c_i) + a'_t(-t_b + \frac{b'_t}{a'_t} t_a) \\ &> h(c_i) + a'_t(-t_b + \frac{t_b}{t_a} t_a) = h(c_i) \end{aligned}$$

which gives the first item.

If $\frac{r_b}{r_a} < \frac{t_b}{t_a}$, then, $h(c_{i-j}) = h(c_i - a_j a - b_j b) = h(c_i) + t_b a_j - t_a b_j$. Thus, we have to prove that $t_b a_j - t_a b_j > t_a + t_b$, which can be rewritten in $\frac{t_b}{t_a}(a_j - 1) > b_j + 1$. On the other hand, the condition 2 can be rewritten in $\frac{r_b}{r_a}(a_j - 1) \geq b_j + 1$ (notice, that, with our convention, $\frac{r_b}{r_a} < \frac{t_b}{t_a}$ implies that $r_a \neq 0$). This ensures that $a_j - 1 > 0$. Thus, since $\frac{r_b}{r_a} < \frac{t_b}{t_a}$, we obtain:

$$\frac{t_b}{t_a}(a_j - 1) > \frac{r_b}{r_a}(a_j - 1) \geq b_j + 1.$$

which is the result.

If $\frac{r_b}{r_a} = \frac{t_b}{t_a}$, we proceed as in the second case to get

$$\frac{t_b}{t_a}(a_j - 1) = \frac{r_b}{r_a}(a_j - 1) \geq b_j + 1.$$

which gives the inequality. On the other hand, $\frac{t_b}{t_a}(a_j - 1) = b_j + 1$ if and only if $t_b a_j - t_a b_j = t_a + t_b$, i.e. $h(c_{i-j}) = h(c_i) + t_a + t_b$. This gives the equivalence.

Corollary 1. *For any configuration c , we have:*

$$h_{min}(c) \leq h_{min}(\delta(c)) \leq h_{max}(\delta(c)) \leq h_{max}(c),$$

and, therefore, $\Delta_h(\delta(c)) \leq \Delta_h(c)$.

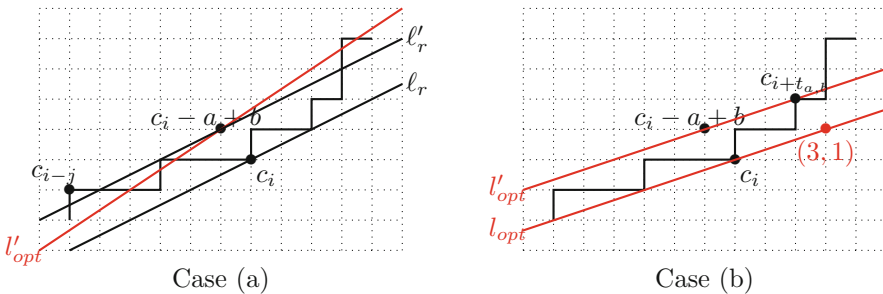


Fig. 2. The main ideas of Lemma 1: the site c_i has a sight of 8, $(r_a, r_b) = (1, 2)$, thus the slope of ℓ_r and ℓ'_r is $\frac{1}{2}$. In the case (a), the line ℓ'_{opt} has a slope of $\frac{2}{3}$ and the site c_{i-j} is over ℓ'_{opt} . In the second case, the line ℓ'_{opt} has a slope of $\frac{1}{3}$ and the site $c_{i+t_{a,b}}$ is over ℓ'_{opt} .

We also have the corollary below, noticing that if a site was flipped in a Christoffel configuration, then either h_{max} would be increased, which is not possible, or h_{min} would be decreased, which is also impossible, by symmetry of the process.

Corollary 2. *Christoffel configurations are stable for any process satisfying the constraints above.*

Strong Thickness Constraint: Assume now that both previous constraints are satisfied, and that, when a flip is done on i , then the new site indexed by i is of maximal height. This may happen when $h(c_i) + t_{a,b} = h_{max}$. If the maximal height appears in i , then the closest indices where the maximal height can eventually also be reached are $i + t_{a,b}$ and $i - t_{a,b}$, because of congruence conditions of Property 1. We want to be sure that our process creates no isolated maximum: if the maximal height h_{max} is reached in i , then i is not an isolated maximum, in the sense of $h(c_{i+t_{a,b}}) = h_{max}$ or $h(c_{i-t_{a,b}}) = h_{max}$. This is useful in the analysis, for energy compensations.

But in the same time, we want to allow a sufficient instability to the process in order to make it move to a better configuration. This is ensured by enforcing the weak thickness constraint as follows (the set $\{1, 2, \dots, s\}$ is denoted by $[s]$):

$$\exists j \in [s] | (a_j - r_a b_j > r_a + r_b) \vee (a_j - r_a b_j = r_a + r_b \wedge \gcd(a_j - 1, b_j + 1) = 1). \tag{3}$$

The strong constraint adds that if all sites c_{i-j} are in the limit line, directed by (r_a, r_b) passing through the site $c_i - a + b$, then there exists a site such that the components of the vector $c_i - a + b - c_{i-j}$ are relatively prime.

If this strong constraint is satisfied, then the weak constraint is automatically satisfied. Nevertheless we prefer to present the process in this way, in order to have a real understanding of the motivations of the rules.

Lemma 2. *Assume that (t_a, t_b) is visible by s . Let c be a configuration, $i \in \{1, 2, \dots, m - 1\}$ and $(w^l(c, i), w^r(c, i)) = (w, w')$. Assume that (w, w') satisfies the three constraints above and $h(c_i) + t_{a,b} = h_{max}(c)$.*

Then $h(c_{i+t_{a,b}}) = h_{max}(c)$ or $h(c_{i-t_{a,b}}) = h_{max}(c)$.

Proof. Lemma 1 directly gives the result when $\frac{r_b}{r_a} > \frac{t_b}{t_a}$, and, from Lemma 1 the hypotheses cannot occur when $\frac{r_b}{r_a} < \frac{t_b}{t_a}$. Thus it remains to study the case when $\frac{r_b}{r_a} = \frac{t_b}{t_a}$ and $h(c_i) + t_{a,b} = h(c_{i-j})$, which ensures that $r_b a_j - r_a b_j = r_a + r_b$, from Lemma 1.

In this case $h(c_{i-j} + a - b) = h(c_{i-j}) - t_b - t_a = h(c_i)$, thus, from Property 1, there exists an integer k such that $c_{i-j} + a - b - c_i = k(t_a, t_b)$, i.e. $(-a_j + 1, -b_j - 1) = k(t_a, t_b)$. We have $-b_j - 1 < 0$ and, from the strong thickness constraint, $\gcd(a_j - 1, b_j + 1) = 1$. Thus, we necessarily have $k = -1$, which gives that $j = a_j + b_j = a_j - 1 + b_j + 1 = t_a + t_b = t_{a,b}$. Thus $h(c_i) + t_{a,b} = h(c_{i-t_{a,b}})$.

4 Analysis

We start by presenting the lemma used to prove time efficiency of our process. Lemma 3 is a classical result about martingales, its proof can be found in [6].

Lemma 3. *Let $k \in \mathbb{N}$ and $\epsilon > 0$. Consider $(c^t)_{t \geq 0}$ a random sequence of configurations, and $E : \mathbb{P} \rightarrow \mathbb{N}$ an energy function. Let, for any c , $\Delta E(c) = E(\delta(c)) - E(c)$, and let $T = \min\{t : E(c^t) = 0\}$ be the random variable which denotes the first time t where $E(c^t) = 0$. Assume that, for any c such that $E(c) > 0$, we conjointly have: $E(c) \leq k$, $\mathbb{E}[\Delta E(c)|c] \leq 0$, and $\text{Prob}\{|\Delta E(c)| \geq 1\} \geq \epsilon$. Then, $\mathbb{E}[T] \leq \frac{kE(c^0)}{\epsilon}$.*

Our strategy consists in using Lemma 3 for an “ad hoc” energy function, that we will define now. Fix a configuration c^0 such that $h_{max}(c^0) \geq t_{a,b}$. The energy $E(c)$ of any configuration c is defined as follows. If $h_{max}(c) \neq h_{max}(c^0)$, then $E(c) = 0$. If $h_{max}(c) = h_{max}(c^0)$, then consider the set $Border^+ = \{i \in \{1, 2, \dots, m - 1\} | \exists i_0 \in \{1, 2, \dots, m - 1\} h(c_{i_0}^0) = h_{max}(c^0) \text{ and } i \equiv i_0 \pmod{(t_{a,b})}\}$.

We recall that if i and j are both elements of $Border^+$, then $i \equiv j \pmod{(t_{a,b})}$. Remark that $n - 1 \leq |Border^+| \leq n$. We define the sets: $Top^+(c) = \{i \in \{1, 2, \dots, m - 1\} | h(c_i) = h_{max}(c^0)\}$, $Down^+(c) = \{(i \in Top^+(c) | i + t_{a,b} \leq m, i + t_{a,b} \notin Top^+(c))\}$, $Up^+(c) = \{(i \in Top^+(c) | i - t_{a,b} \geq 0, i - t_{a,b} \notin Top^+(c))\}$. Notice that we have $Top^+(c) \subseteq Border^+$. The energy $E(c)$ of the configuration c is the sum:

$$E(c) = 2|Top^+(c)| + |Down^+(c)| + |Up^+(c)|$$

Proposition 1. *When $h_{max}(c^0) \geq t_{a,b}$, the energy defined above satisfies the hypotheses of Lemma 3 with $\epsilon = \frac{1}{m-1}$, and $k = 3n$*

Proof. (sketch) One easily sees that $E(c) < 3n$ since $|Top^+(c)| \leq n$, $|Down^+(c)| + |Up^+(c)| \leq n - 1$ and at least one equality must be strict. This gives $E(c) \leq 3n$.

Make a partition $P^+(c)$ of $Border^+(c)$ in subsets of at most three consecutive elements in such a way that, for each $i \in Down^+(c)$, integers i and $i + t$ are in the same subset, and for each $i \in Up^+(c)$ such that $i - 2t_{a,b} \notin Top^+(c)$, then integers i and $i - t_{a,b}$ are in the same subset. A straightforward case by case analysis gives that or each subset S the total contribution of elements of S to the value of $\mathbb{E}[\Delta E(c)|c]$ is not positive. Thus, we get $\mathbb{E}[\Delta E(c)|c] \leq 0$.

The fact that $\text{Prob}\{|\Delta E(c)| \geq 1\} \geq \frac{1}{m-1}$ is a direct consequence of Lemma 4 below. By definition, Top^+ is not empty. The site c_i of Top^+ of lowest index is active. (notice that we need the hypothesis : $h_{max}(c^0) \geq t_{a,b}$ to ensure it, in the case when $i < t_{a,b}$), and when i is randomly chosen, with probability $\frac{1}{m-1}$, the energy decreases from at least 1 unit (actually 2 units, except when $i < t_{a,b}$ and $i + t_{a,b} \in Top^+(c)$), or $i > m - t_{a,b}$ and $i - t_{a,b} \in Top^+(c)$.

Lemma 4. *Let c be a configuration. Assume that there exists $i \leq m - s$ such that $h(c_i) = h_{max}(c)$ (respectively $h(c_i) = h_{min}(c)$), and there exists j such that $0 < j \leq t_{a,b}$ and $h(c_{i-j}) + t_{a,b} \leq h_{max}(c)$ (respectively $h(c_{i-j}) - t_{a,b} \geq h_{min}(c)$). Then, the site c_i is active in c .*

The proof is done by a local analysis.

4.1 Results

Nonnegative Configurations. We say that a configuration is nonnegative if for each $i \in \{0, 1, \dots, m\}$ we have $h(c_i) \geq 0$.

Theorem 1. If (t_a, t_b) is visible by s , and the configuration c^0 is nonnegative, then the random process is a coalescence process in $O(n^4)$ time units in average. The configuration reached is the unique Christoffel configuration $c_{[0, t_a, b-1]}$ such that $h_{max}(c_{[0, t_a, b-1]}) = t_{a,b} - 1$ and $h_{min}(c_{[0, t_a, b-1]}) = 0$.

Proof. We can decompose $T = \sum_{j=t_a, b}^{2n-1} T_j$ where T_j is the time to get a configuration c such that $h_{max}(c) \geq j - 1$ from a configuration c' such that $h_{max}(c') \geq j$. From Proposition 1, we have $\mathbb{E}[T_j] \leq (2n - 1)^2(m - 1)$, thus $\mathbb{E}[T] \leq (2n - 1)^3(m - 1)$. Moreover, from Corollary 1 we have, for any t , $h_{min}(c^t) \geq 0$. Thus, after time T , we get a configuration c such that, $h_{max}(c) \leq t_{a,b} - 1$ and $h_{min}(c) \geq 0$ which ensures that $c = c_{[0, t_a, b-1]}$. We know that $c_{[0, t_a, b-1]}$ is stable, from Corollary 2.

The General Case with Fixed Endpoints. If we work with two energies, one as described above, related to h_{max} , and one symmetric, related to h_{min} , one gets, in a similar way:

Theorem 2. If (t_a, t_b) is visible by s , then, with any initial configuration, the random process almost surely reaches a configuration c such that: $-t_{a,b} + 1 \leq h_{min}(c) < h_{max}(c) \leq t_{a,b} - 1$. The time T necessary to reach such a configuration is $O(n^4)$ in average.

Notice that our arguments fail to continue to decrease the thickness, because of difficulties at the boundary. For $h_{max}(c) \leq t_{a,b} - 1$, when the lowest element i of $Top^+(c)$ is such that $i \leq t_{a,b}$, we can not ensure that i is active.

Thus Theorem 2 is partially satisfying: we reach a set of configurations which are only partially stable. Sites c_i with $i \equiv 0 \pmod{t_{a,b}}$ are no more active and are on the ideal line of equation $-t_b x + t_a y = 0$. But some other sites can remain active. Nevertheless, these sites only have the freedom to oscillate around the ideal line, between the two positions which are the closest ones to the ideal line, *i.e.* the positions of lowest positive height, and of largest negative height.

It is not possible to always get the optimal thickness with our algorithm. For example, for $s = 5$ and $(t_a, t_b) = (3, 2)$, consider the configuration c associated word $(ba^2ba)^{n-1}ba^3b$. We have $h(c_{m-1}) = h_{min}(c) = -3$ and $h(c_1) = h_{max}(c) = 3$. Moreover, with Lemma 1, one can easily see that, for any integer $t > 0$, $h(\delta^t(c)_1) = 3$ and $h(\delta^t(c)_{m-1}) = -3$.

The General Periodic Case. Notice that configurations can be seen as cycles by identifying site c_0 and site c_m . We call it the *cyclic model*. Formally, instead of considering sites as elements of \mathbb{Z}^2 , they are considered as elements of the quotient space $\mathbb{Z}^2/n(t_a, t_b)\mathbb{Z}$. This makes two main differences: the site c_0 can be possibly active, and for each configuration c and each index i , we have $|w^l(c, i)| = |w^r(c, i)| = s$, so the sight constraint becomes irrelevant.

Using a very light modification of the energy function (adding two units for the energy when $Top^+(c) = Border^+$), we obtain the following result.

Theorem 3. In the cyclic model, if (t_a, t_b) is visible by s , from any origin configuration c^0 , the random process is a coalescence process whose coalescence time T is $O(n^4)$ in average.

Theorem 4 (Impossibility Result). Consider any local rule δ of sight s for which Christoffel configurations of slope $\frac{1}{s+1}$ are stable.

Then, for any $k > 0$, there exists a configuration c , linking $(0, 0)$ to $(2k(s+1), 2k)$, such that $\Delta_h(c) \geq k$ and c is stable for δ .

Proof Consider the words $w = a^{s+1}b$, $w' = a^s b$, $w'' = a^{s+2}b$, the configuration c corresponding to w^{2k} and c' corresponding to $w(w'')^{k-1}(w')^{k-1}w$. The configuration c corresponds to a Christoffel word where all letters b are separated by at least $s + 1$ letters a . Then, if c is stable under the local rule δ , then $\delta(a^s, ba^{s-1}) = \delta(ba^{s-1}, a^s) = 0$.

The equalities above ensure that if a letter b is surrounded by s letters a in each of its sides, then this letter cannot be involved in a flip induced by the process δ . It follows that c' is stable for δ , since in configuration c' , all letters b are separated by at least s letters a .

5 Conclusion and Open Questions

In this part, we start by presenting the improvements which can be done to this paper. Then we focus on the possible extensions and applications of our work.

We think that the rule introduced here is optimal in terms of sight and convergence speed in our setting. Nevertheless, we think that our analysis is not optimal and we conjecture that our random process converges in $O(n^3)$. Our analysis considers only the sites of maximal height and forgets about a lot of useful updates which are done in parallel. Also, we think that our rule for synchronizing the endpoints is not optimal in time and sight. We conjecture that both endpoints can be synchronized in polynomial time according to n, s and $t_{a,b}$ with agents whose sight is $2s$ (which is the minimum to see two potential periods, when there is no visibility in one side) at the endpoints.

Another interesting question is to generalize our process to dimensions greater than two, *i.e.* to an alphabet with more than two letters for the language theory version of this problem. In ongoing works, our process is working well experimentally in greater dimensions, if it is given a big enough sight,

but we are not able yet to prove that there is no interlocking between the letters. This extension is interesting for two applications of our work.

The first application is for studying a model of cooling processes in crystallography [2, 3, 7]. In this paper, we study in fact a “simple” case where two kinds of atoms are disposed on a line and these atoms want to diminish the interactions with the atoms of the same kind. Increasing the dimension of this problem corresponds to consider more than two kinds of atoms. All these studies aims to present and analyze a cooling model for a $2D$ aperiodic tiling like Penrose tiling. Aperiodic tilings correspond to quasicrystal and actually fabricating a quasicrystal of large size is not possible because the cooling process is not well understood. The result presented here is the first which does not require to have the same proportions of the different kinds of atoms.

The second application would be to generalize the density classification problem [10]. In this problem, we consider a one dimensional chain of agents. There are two states and each agent can memorize only one state. Using a distributed algorithm, agents must determine the majority state in the initial configuration while storing only one state by agent. Recently Fatès [5] solved this problem with any arbitrary precision using a probabilistic dynamics. We think that our result can be used to generalize the density classification to densities different than $\frac{1}{2}$. Moreover, if our process can be generalized to more than two states, then we may be able to solve the density classification problem, with more than two states.

References

1. Berstel, J.: Sturmian and episturmian words. In: Bozopalidis, S., Rahonis, G. (eds.) CAI 2007. LNCS, vol. 4728, pp. 23–47. Springer, Heidelberg (2007)
2. O. Bodini, T. Fernique, and D. Regnault. Quasicrystallization by stochastic flips. In: Proceedings of Aperiodics 2009 Journal of Physics, vol. 226(012022) (2010)
3. Bodini, O., Fernique, T., Regnault, D.: Stochastic flip of two-letters words. In: Proceedings of ANALCO2010, pp. 48–55. SIAM (2010)
4. Dynia, M., Kutylowski, J., Lorek, P., auf der Heide, F.M.: Maintaining communication between an explorer and a base station. In: Pan, Y., Rammig, F.J., Schmeck, H., Solar, M. (eds.) BICC 2006. IFIP. Springer, Heidelberg (2006)
5. Fatès, N.: Stochastic cellular automata solve the density classification problem with an arbitrary precision. In: Proceedings of STACS 2011, pp. 284–295 (2011)
6. Fatès, N., Morvan, M., Schabanel, N., Thierry, É.: Fully asynchronous behavior of double-quiescent elementary cellular automata. *Theor. Comput. Sci.* **362**(1–3), 1–16 (2006)
7. Fernique, T., Regnault, D.: Stochastic flip on dimer tilings. In: Proceedings of AofA2010, DMTCS, vol. AM, pp. 207–220 (2010)
8. Henley, C.L.: Random tiling models. In: Quasicrystal, A State of the Art. World Scientific (1991)
9. Burton, Jr., J.K., Henley, C.L.: A constrained potts antiferromagnet model with an interface representation. *J. Phys. A* **30**, 8385–8413 (1997)
10. Packard, N.H.: Adaptation toward the edge of chaos. In: Dynamic Patterns in Complex Systems, pp. 293–301. World Scientific, Singapore (1988)
11. Thurston, W.P.: Conways tiling groups. *Am. Math. Mon.* **97**, 757–773 (1990)

Equations and Coequations for Weighted Automata

Julian Salamanca¹(✉), Marcello Bonsangue^{1,2}, and Jan Rutten^{1,3}

¹ CWI Amsterdam, Amsterdam, The Netherlands
salamanc@cwi.nl

² LIACS - Leiden University, Leiden, The Netherlands

³ Radboud University Nijmegen, Nijmegen, The Netherlands

Abstract. We study weighted automata from both an algebraic and a coalgebraic perspective. In particular, we consider equations and coequations for weighted automata. We prove a duality result that relates sets of equations (congruences) with (certain) subsets of coequations. As a consequence, we obtain two equivalent but complementary ways to define classes of weighted automata. We show that this duality cannot be generalized to linear congruences in general but we obtain partial results when weights are from a field.

1 Introduction

Weighted automata are a generalization of non-deterministic automata introduced by Schützenberger [14]. Every transition is associated with an input letter from an alphabet A and a weight expressing the cost (or probability, time, resources needed) of its execution. This weight is typically an element of a semiring. The multiplication of the semiring is used to accumulate the weight of a path by multiplying the weights of each transition in the path, while the addition of the semiring computes the weight of a string w by summing up the weights of the paths labeled with w [11]. In this way, the behaviour of weighted automata is given in terms of formal power series, i.e. functions assigning a weight to each finite string w over A .

Weighted automata may have a non-deterministic behaviour because different transitions from the same state may be labeled by the same input letter, with possibly different weights. However, they can be determinized by assigning a linear structure to the state-space using a generalization of the powerset construction for non-deterministic automata [5]. As such, determinized weighted automata are typically infinite-state, but determinization allows us to study weighted automata both from an algebraic perspective and a coalgebraic one. From the algebraic perspective, a (determinized) weighted automaton is just an algebra with a unary operation for each input symbol, whereas coalgebraically,

J. Salamanca—The research of this author is funded by the Dutch NWO project 612.001.210.

a weighted automaton is a deterministic transition system with output weights associated to each state.

In this context, and building on the work by [3] on ordinary deterministic automata and on the duality between reachability and observability [2, 4, 6], we study equations and coequations for weighted automata. In general, equations characterize classes of algebras called varieties [7], whereas coequations characterize classes of coalgebras, so-called covarieties [13]. Using the algebraic perspective, an equation for a weighted automaton is just a pair of words (u, v) . Satisfaction becomes reachability: a weighted automaton satisfies an equation (u, v) if from any state, the linear combination of states reached after reading u is the same as the one reached after reading v [3].

Dually, the coalgebraic perspective allows us to define coequations for weighted automata as subsets (or predicates) of power series. A weighted automaton M satisfies a coequation S if for any function associating an output weight to each state, the behaviour of M from any initial state is a power series in S .

Our main result is a duality between sets of equations called congruences and sets of coequations called closed subsystems. More precisely, we prove that the classes of weighted automata defined by congruences are in a one-to-one correspondence with classes of weighted automata defined by closed subsystems. This allows, for example, to give equational characterizations to some specific subsets of power series and, vice versa, to define the least subset of power series specified by a set of equations.

In the second part of the paper, we retain the linear structure of the transitions of the determinized weighted automata, allowing for a more general kind of equations, called linear. In general, a full duality result does not hold anymore, but when the weights come from a field, we still have one direction of it: a variety defined by a linear congruence can be recovered from a corresponding covariety. As an example, we show that linear congruences (under certain conditions) are finitely generated by a set of equations, using the fact that by Hilbert basis theorem [10, VIII, Theorem 4.9], linear congruences correspond to ideals of polynomials.

We will proceed as follows. After some mathematical preliminaries, in Sect. 3 we show how to construct the set of equations and coequations for a given weighted automaton. In Sect. 4, we give the duality result between congruences and closed subsystems. In Sect. 5, we move from Set to vector spaces and linear equations. We conclude, in Sect. 6, with a discussion of related work.

2 Preliminaries

In this section, we will define the main concepts and notation used in this paper. Given two sets X and Y we define $Y^X = \{f \mid f : X \rightarrow Y\}$. For a function $f \in Y^X$, we define the *kernel* and the *image* of f by

$$\ker(f) = \{(x_1, x_2) \in X \times X \mid f(x_1) = f(x_2)\} \quad \text{Im}(f) = \{f(x) \mid x \in X\}.$$

For any set A we denote by A^* the free monoid on A , its identity element will be denoted by ϵ . Given an element $f \in X^{A^*}$ and $w \in A^*$, we define the *right*

derivative f_w of f with respect to w and the left derivative ${}_w f$ of f with respect to w as the elements ${}_w f, f_w \in X^{A^*}$ such that for every $u \in A^*$,

$$f_w(u) = f(wu), \text{ and } {}_w f(u) = f(uw).$$

Let A be an alphabet (not necessarily finite), a *deterministic automaton* on A is a pair (X, α) where $\alpha : X \times A \rightarrow X$ is a function. We can add an initial state $x_0 \in X$ or an output function $c : X \rightarrow O$ to get a *pointed automaton* (X, x_0, α) or a *Moore automaton* (X, c, α) with outputs in O , respectively. For any $x \in X$ and $u \in A^*$ we define $u(x)$ inductively as $\epsilon(x) = x$ and $wa(x) = \alpha(w(x), a)$.

We have that pointed automata are F -algebras for the endofunctor F on Set given by $F(X) = 1 + (A \times X)$. By using the correspondence $\alpha : X \times A \rightarrow X \Leftrightarrow \alpha' : X \rightarrow X^A$, given by $\alpha(x, a) = \alpha'(x)(a)$, Moore automata with outputs in O are G -coalgebras for the endofunctor G on Set given by $G(X) = O \times X^A$. The initial F -algebra is (A^*, ϵ, τ) , where the transition function τ is concatenation, that is $\tau(w, a) = wa$, for all $w \in A^*$ and $a \in A$, and for any pointed automaton (X, x_0, α) the unique F -algebra morphism $r_{x_0} : (A^*, \epsilon, \tau) \rightarrow (X, x_0, \alpha)$ is given by $r_{x_0}(w) = w(x_0)$. Dually, the final G -coalgebra is $(O^{A^*}, \hat{\epsilon}, \hat{\tau})$, where $\hat{\epsilon}(f) = f(\epsilon)$ and $\hat{\tau}(f)(a) = f_a$, and for any Moore automaton (X, c, α) with outputs in O , the unique G -coalgebra morphism $o_c : (X, c, \alpha) \rightarrow (O^{A^*}, \hat{\epsilon}, \hat{\tau})$ is given by $o_c(x) = \lambda w. c(w(x)) \in O^{A^*}$.

A *semiring* \mathbb{S} is an algebra $\mathbb{S} = (S, +, \cdot, 0, 1)$, where $+$ and \cdot are binary operations and 0 and 1 are nullary operations, such that $(S, +, 0)$ is a commutative monoid, $(S, \cdot, 1)$ is a monoid, \cdot distributes over $+$ on the left and on the right, and $0 \cdot s = s \cdot 0 = 0$ for every $s \in S$. Given a semiring \mathbb{S} , a *semimodule* over \mathbb{S} , or \mathbb{S} -*semimodule*, is a commutative monoid $R = (R, +, 0)$ together with an \mathbb{S} -left-action $\cdot : S \times R \rightarrow R$ such that

$$\begin{aligned} (s + s') \cdot r &= s \cdot r + s' \cdot r & 0 \cdot r &= 0 & 1 \cdot r &= r \\ s \cdot (r + r') &= s \cdot r + s \cdot r' & s \cdot 0 &= 0 & s \cdot (s' \cdot r) &= (s \cdot s') \cdot r \end{aligned}$$

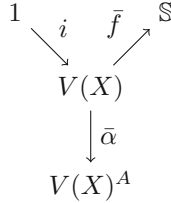
for any $s, s' \in S$ and $r, r' \in R$. We will often write sr instead of $s \cdot r$. For an alphabet A and a semiring \mathbb{S} , elements in \mathbb{S}^{A^*} are called *power series*. Given a set X , the free \mathbb{S} -semimodule on the generators X , denoted by $V(X)$, is the \mathbb{S} -semimodule whose underlying set is $V(X) = \{\phi \in \mathbb{S}^X \mid \text{supp}(\phi) \text{ is finite}\}$, where $\text{supp}(\phi)$, the *support* of ϕ , is defined as $\text{supp}(\phi) = \{x \in X \mid \phi(x) \neq 0\}$. Addition in $V(X)$ is component-wise, $0 \in V(X)$ is the constant function with 0 as its value, and the action of \mathbb{S} over $V(X)$ is multiplication of a constant by a function. For $\phi \in V(X)$ we have the correspondence $\phi \Leftrightarrow s_1 \cdot x_1 + \dots + s_n \cdot x_n$, where $\text{supp}(\phi) = \{x_1, \dots, x_n\}$ and $\phi(x_i) = s_i, i = 1, \dots, n$. Notice that we are using formal sums here. According to this correspondence there is a copy of X in $V(X)$, namely $x \mapsto 1 \cdot x$; in this case, $1 \cdot x$ will be simply denoted as x .

A *linear map* between \mathbb{S} -semimodules S_1 and S_2 is a function $h : S_1 \rightarrow S_2$ such that for any $x, y \in S_1$ and $c, d \in \mathbb{S}$, $h(cx + dy) = ch(x) + dh(y)$. Let \mathbb{S}

be an \mathbb{S} -semimodule, X a set, and $f : X \rightarrow S$ a function. We define the *linear extension of f* as the linear map $\bar{f} : V(X) \rightarrow S$ given by

$$\bar{f}(c_1 \cdot x_1 + \cdots + c_n \cdot x_n) = c_1 f(x_1) + \cdots + c_n f(x_n).$$

A *weighted automaton* with input alphabet A and weights over a semiring \mathbb{S} is a pair (X, α) , where X is a set (not necessarily finite) and $\alpha : X \rightarrow V(X)^A$ is a function. We can add an initial state $i : 1 \rightarrow V(X)$ and/or a final state function, or colouring, $f : X \rightarrow \mathbb{S}$, yielding the following situation:



Notice that if we remove \bar{f} we have a pointed automaton, and if we remove i we have a Moore automaton with outputs in \mathbb{S} . In particular, every weighted automaton (X, α) gives rise to a deterministic automaton $(V(X), \bar{\alpha})$.

An *equation* is a pair $(u, v) \in A^* \times A^*$, sometimes also denoted by $u = v$. Given a weighted automaton (X, α) and an equation $(u, v) \in A^* \times A^*$, we define $(X, \alpha) \models (u, v)$; and say (X, α) satisfies the equation (u, v) , as follows:

$$(X, \alpha) \models (u, v) \Leftrightarrow \forall \phi \in V(X) \ u(\phi) = v(\phi),$$

recall the definition of $u(x)$ at the beginning of this section. For any set of equations $E \subseteq A^* \times A^*$ we write $(X, \alpha) \models E$ if $(X, \alpha) \models (u, v)$ for every $(u, v) \in E$. Observe that since $\bar{\alpha}$ is the linear extension of α , the condition $\forall \phi \in V(X) \ u(\phi) = v(\phi)$ is equivalent to $\forall x \in X \ u(x) = v(x)$. In other words, $u = v$ is satisfied if the *linear combination* of states in $V(X)$ reached after u from every $x \in X$ is equal to that reached after v .

A set of *coequations* is a subset $D \subseteq \mathbb{S}^{A^*}$. We define $(X, \alpha) \models D$; and say (X, α) satisfies the set of coequations D , as follows:

$$(X, \alpha) \models D \Leftrightarrow \forall f \in \mathbb{S}^X, \phi \in V(X) \ o_{\bar{f}}(\phi) \in D$$

The power series $o_{\bar{f}}(\phi)$ is the *behaviour* of the state ϕ in (X, α) with respect to the colouring f . According to this, $(X, \alpha) \models D$ means that D includes all the possible behaviours for the automaton (X, α) .

An equivalence relation C on A^* is a *congruence* on A^* if for any $t, u, v, w \in A^*$, $(t, v) \in C$ and $(u, w) \in C$ imply $(tu, vw) \in C$. If C is a congruence on A^* , the *congruence quotient* A^*/C has a pointed automaton structure $A^*/C = (A^*/C, [\epsilon], [\tau])$ with transition function given by $[\tau]([w], a) = [\tau(w, a)] = [wa]$, which is well defined since C is a congruence.

3 Free and Cofree Construction for Weighted Automata

In this section we will show how to construct the maximum set of equations and the minimum set of coequations satisfied by a weighted automaton (X, α) (thereby generalizing the approach of [3, Sect.5]). We use the notation \mathbf{free}_w and \mathbf{cofree}_w to distinguish between the free and cofree construction for weighted automata from the construction \mathbf{free} and \mathbf{cofree} for deterministic automata defined in [3], respectively.

To get the maximum set of equations of (X, α) we are going to construct the pointed deterministic automaton $\mathbf{free}_w(X, \alpha)$ by taking the following steps:

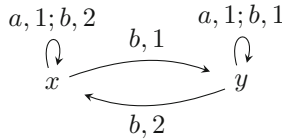
1. Define the pointed deterministic automaton $\prod(X, \alpha) = (\prod_{x \in X} V(X), \Delta, \hat{\alpha})$ where $\hat{\alpha}$ is the product of $\bar{\alpha} |X|$ times, that is $\hat{\alpha}(\theta)(a)(x) = \bar{\alpha}(\theta(x))(a)$, and $\Delta \in \prod_{x \in X} V(X)$ is given by $\Delta(x) = x$. Then, by initiality of $A^* = (A^*, \epsilon, \tau)$, we get a unique F -algebra morphism $r_\Delta : A^* \rightarrow \prod(X, \alpha)$.
2. Define $\mathbf{free}_w(X, \alpha)$ and $\mathbf{Eq}_w(X, \alpha)$ as

$$\mathbf{free}_w(X, \alpha) := A^* / \ker(r_\Delta) \text{ and } \mathbf{Eq}_w(X, \alpha) := \ker(r_\Delta)$$

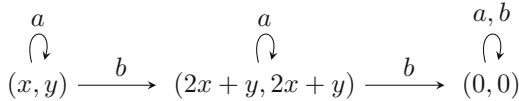
By construction, we have the following theorem.

Theorem 1. $\mathbf{Eq}_w(X, \alpha)$ is the maximum set of equations satisfied by (X, α) .

Example 2. Consider the weighted automaton with input alphabet $A = \{a, b\}$ and weights on the semiring (field) \mathbb{Z}_3 given by the following diagram:



According to the definition $\mathbf{free}_w(X, \alpha) = A^* / \ker(r_\Delta) \cong \text{Im}(r_\Delta)$, so in order to construct $\mathbf{free}_w(X, \alpha)$ we only need to construct the reachable part of $\prod(X, \alpha)$ from the state $\Delta = (x, y)$. By doing that we get the automaton:



Thus $\mathbf{Eq}_w(X, \alpha)$ is the congruence generated by $\{a = \epsilon, bb = bbb\}$ and $\mathbf{free}_w(X, \alpha)$ is the automaton with states $[\epsilon] = a^*$, $[b] = a^*ba^*$, and $[bb] = a^*ba^*b(a + b)^*$. □

Now we will show how to get the minimum set of coequations satisfied by (X, α) . First some notation: for any family of sets $\{X_i\}_{i \in I}$ we denote by $\coprod_{i \in I} X_i$ the disjoint union (coproduct in Set) of the family which is given by $\coprod_{i \in I} X_i = \bigcup_{i \in I} \{i\} \times X_i$. We will construct the Moore automaton $\mathbf{cofree}_w(X, \alpha)$ by taking the following steps:

1. Define the Moore automaton $\coprod(X, \alpha) = (\coprod_{f \in \mathbb{S}^X} V(X), \Phi, \tilde{\alpha})$ where $\tilde{\alpha}$ and Φ are given by $\tilde{\alpha}(f, \phi)(a) = (f, a(\phi))$ and $\Phi(f, \phi) = \tilde{f}(\phi)$. Then, by finality of $\mathbb{S}^{A^*} = (\mathbb{S}^{A^*}, \hat{\epsilon}, \hat{\tau})$, we get a unique G -coalgebra morphism $o_\Phi : \coprod(X, \alpha) \rightarrow \mathbb{S}^{A^*}$.
2. Define $\mathbf{cofree}_w(X, \alpha)$ and $\mathbf{coEq}_w(X, \alpha)$ as

$$\mathbf{cofree}_w(X, \alpha) = \mathbf{coEq}_w(X, \alpha) := \text{Im}(o_\Phi).$$

Similarly as in the case of equations we have the following theorem.

Theorem 3. $\mathbf{coEq}_w(X, \alpha)$ is the minimum set of coequations satisfied by (X, α) .

4 Duality Between Equations and Coequations

In this section, we will use the free and cofree construction given in [3, Sect. 5], to show a duality result between equations and coequations (here the we are going to use the semiring \mathbb{S} as a set of colours on the coalgebraic side). In the sequel \mathbb{S} will be a fixed semiring.

Proposition 4. For every congruence quotient A^*/C ,

$$\mathbf{cofree}(A^*/C) = \{f \in \mathbb{S}^{A^*} \mid C \subseteq \ker(f)\}.$$

Observe that the previous proposition is a generalization of [3, Proposition 14]. As a consequence, we have:

Theorem 5. For any congruence quotient A^*/C , $\mathbf{free} \circ \mathbf{cofree}(A^*/C) = A^*/C$.

Now we will define one of the main concepts that will lead us to the duality. A subset $S \subseteq \mathbb{S}^{A^*}$ is called a *closed subsystem* if

- (i) S is closed under left and right derivatives.
- (ii) $\mathbf{B}(S) \stackrel{\text{def}}{=}} (\{\text{supp}(f) \mid f \in S\}, \cap, ()', A^*)$ is a complete atomic Boolean algebra.
- (iii) Let $\text{At}(\mathbf{B}(S))$ be the set of atoms of $\mathbf{B}(S)$, then

$$S = \left\{ f \in \mathbb{S}^{A^*} \mid \forall P \in \text{At}(\mathbf{B}(S)) \ f \upharpoonright_P \text{ is constant} \right\}.$$

Notice that condition i) implies that in fact S is a subsystem of \mathbb{S}^{A^*} , i.e. a subcoalgebra. For the case of the Boolean semiring, that is $\mathbb{S} = \mathbb{B}$, the notion of closed subsystem coincides with that of preformation of languages defined in [3], which was one of the reasons and motivations to state the previous definition in its final form. We can think of closed subsystems as \mathbb{S} -colourings of the atoms of $\mathbf{B}(S)$, according to condition iii), which in the case of preformations of languages are only 2-colourings, but we also need those colourings to be well-behaved in the sense of being closed under left and right derivatives to get a subsystem of the final coalgebra \mathbb{S}^{A^*} , and also to induce a congruence defining the system in the following sense.

Theorem 6. *Let S be a subset of \mathbb{S}^{A^*} . S is a closed subsystem if and only if $S = \mathbf{cofree}(A^*/C)$ for some congruence quotient A^*/C .*

Observe that by Theorem 5, the congruence C of the previous theorem is unique.

Example 7. Let $S \subseteq \mathbb{S}^{A^*}$ be the set $S = \{f \in \mathbb{S}^{A^*} \mid \forall w \in A^* f(w) = f(b^{|w|_b})\}$, where $|w|_b$ is the number of b 's in the word w . Then one can verify that S is a closed subsystem in which $\text{At}(\mathbf{B}(S)) = \{a^*, a^*ba^*, a^*(ba^*)^2, a^*(ba^*)^3, \dots\}$ and $S = \mathbf{cofree}(A^*/C)$, if we take for C the congruence generated by $\{a = \epsilon\}$. \square

Corollary 8. *For any closed subsystem S of \mathbb{S}^{A^*} , $\mathbf{cofree} \circ \mathbf{free}(S) = S$.*

Example 9. Consider the congruence quotient $A^*/C = \{[\epsilon], [b], [bb]\}$ from Example 2, that is $A = \{a, b\}$ and C is the congruence generated by $\{a = \epsilon, bb = bbb\}$. If $\mathbb{S} = \mathbb{Z}_3$, then by Proposition 4 we know that

$$\mathbf{cofree}(A^*/C) = \left\{ f \in (\mathbb{Z}_3)^{A^*} \mid \forall [w] \in A^*/C, f|_{[w]} \text{ is constant} \right\},$$

that is $\mathbf{cofree}(A^*/C)$ has 27 elements. For any partition $\mathcal{A} = \{[w_i] \mid 1 \leq i \leq n\}$ of A^* and $c_i \in \mathbb{S}$ we define the function $c_1\chi_{[w_1]} + \dots + c_n\chi_{[w_n]} \in \mathbb{S}^{A^*}$ as $(c_1\chi_{[w_1]} + \dots + c_n\chi_{[w_n]})(w) = c_i$ if and only if $w \in [w_i]$, which is well-defined since \mathcal{A} is a partition of A^* . Using the previous notation, we have that every element in $\mathbf{cofree}(A^*/C)$ is of the form $c_1\chi_{[\epsilon]} + c_2\chi_{[b]} + c_3\chi_{[bb]}$, $c_i \in \mathbb{Z}_3$. Here are some examples:

- (i) Consider the colouring $c : A^*/C \rightarrow \mathbb{Z}_3$ given by $c([\epsilon]) = 0$, $c([b]) = 1$, and $c([bb]) = 2$, then we get that $o_c([b]) \in (\mathbb{Z}_3)^{A^*}$ is given by

$$o_c([b])(w) = \begin{cases} 1 & \text{if } w \in a^*, \\ 2 & \text{otherwise.} \end{cases}$$

That is, $o_c([b]) = \chi_{[\epsilon]} + 2\chi_{[b]} + 2\chi_{[bb]}$.

- (ii) For any $c_i \in \mathbb{Z}_3$, the reader can easily verify the following identities for left derivatives

$$\begin{aligned} a(c_1\chi_{[\epsilon]} + c_2\chi_{[b]} + c_3\chi_{[bb]}) &= c_1\chi_{[\epsilon]} + c_2\chi_{[b]} + c_3\chi_{[bb]} \\ b(c_1\chi_{[\epsilon]} + c_2\chi_{[b]} + c_3\chi_{[bb]}) &= c_2\chi_{[\epsilon]} + c_3\chi_{[b]} + c_3\chi_{[bb]}. \end{aligned}$$

\square

Next we define the category \mathcal{C} of *congruence quotients* and the category \mathcal{K} of *closed subsystems*, for a fixed semiring \mathbb{S} , as follows:

$$\begin{aligned} \text{objects}(\mathcal{C}) &= \{(A^*/C, [\tau]) \mid A^*/C \text{ is a congruence quotient}\} \\ \text{arrows}(\mathcal{C}) &= \{e : A^*/C \rightarrow A^*/D \mid e \text{ is an epimorphism}\} \\ \text{objects}(\mathcal{K}) &= \{(K, \alpha) \mid K \text{ is a closed subsystem of } \mathbb{S}^{A^*}\} \\ \text{arrows}(\mathcal{K}) &= \{m : K \rightarrow K' \mid m \text{ is a monomorphism}\} \end{aligned}$$

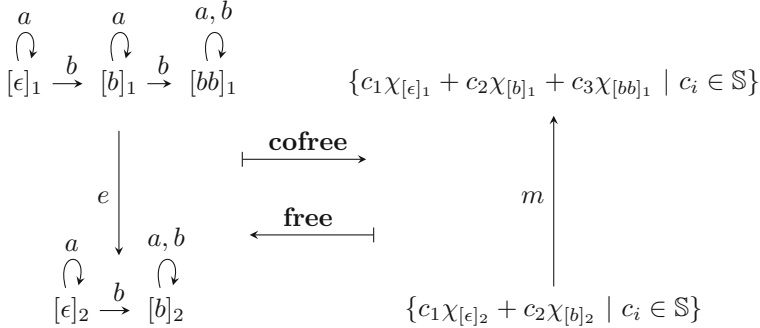
By Theorem 5 and Corollary 8, we have the following.

Theorem 10. $\mathbf{cofree} : \mathcal{C} \cong \mathcal{K}^{op} : \mathbf{free}$

Example 11. Let C_1 be the congruence generated by $\{a = \epsilon, bb = bbb\}$ and let C_2 be the congruence generated by $\{a = \epsilon, b = bb\}$. Clearly we have that $C_1 \subseteq C_2$, $A^*/C_1 = \{[\epsilon]_1, [b]_1, [bb]_1\}$, and $A^*/C_2 = \{[\epsilon]_2, [b]_2\}$ where

$$\begin{aligned} [\epsilon]_1 &= [\epsilon]_2 = a^* & [b]_1 &= a^*ba^* \\ [bb]_1 &= a^*ba^*b(a+b)^* & [b]_2 &= a^*b(a+b)^* = [b]_1 \cup [bb]_1. \end{aligned}$$

By the duality, we have the following situation:



where the epimorphism $e : A^*/C_1 \rightarrow A^*/C_2$ is given by $e([w]_1) = [w]_2$, and the monomorphism $m : \mathbf{cofree}(A^*/C_2) \rightarrow \mathbf{cofree}(A^*/C_1)$ is given by $m(c_1\chi_{[\epsilon]_2} + c_2\chi_{[b]_2}) = c_1\chi_{[\epsilon]_1} + c_2\chi_{[b]_1} + c_2\chi_{[bb]_1}$. \square

There is also the following consequence of the duality, which basically tells us that we can either work with congruences or closed subsystems in weighted automata.

Theorem 12. Let $S \subseteq \mathbb{S}^{A^*}$ be a closed subsystem, C a congruence on A^* , and (X, α) a weighted automata. Then

- (i) $(X, \alpha) \models C$ if and only if $(X, \alpha) \models \mathbf{coEq}(A^*/C)$.
- (ii) $(X, \alpha) \models S$ if and only if $(X, \alpha) \models \mathbf{Eq}(S)$.

Example 13. (Example 7 continued) Let $A = \{a, b\}$. We showed the correspondence between the congruence C on A^* generated by $\{a = \epsilon\}$ and the closed subsystem $S \subseteq \mathbb{S}^{A^*}$ given by $S = \{f \in \mathbb{S}^{A^*} \mid \forall w \in A^* f(w) = f(b^{|w|}b)\}$, that is $\mathbf{cofree}(A^*/C) = S$ or, equivalently, $\mathbf{free}(S) = A^*/C$. In this case, we have, for a weighted automaton (X, α) on A

$$(X, \alpha) \models C \Leftrightarrow (X, \alpha) \models S \Leftrightarrow \forall x \in X \alpha(x, a) = x. \quad \square$$

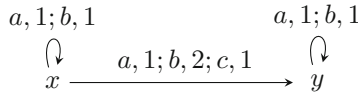
5 Linear Equations and Coequations

In this section, we work with a more general kind of equations and a more general kind of automata. In the previous sections we defined equations to be pairs $(u, v) \in A^* \times A^*$ and we said that an automaton (X, α) with weights in \mathbb{S} satisfies (u, v) if for any $x \in X$, $u(x) = v(x)$. Since $u(x) \in V(X)$ (the free \mathbb{S} -semimodule generated by X) for any $u \in A^*$, it makes sense to define expressions like

$$(s_1w_1 + \dots + s_nw_n)(x) := s_1w_1(x) + \dots + s_nw_n(x)$$

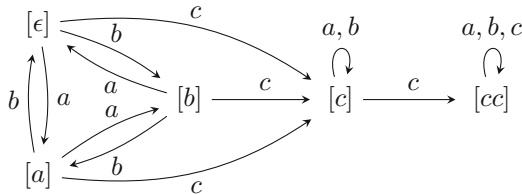
for elements $s_i \in \mathbb{S}$ and $w_i \in A^*$, that is, for any $\varphi = \sum_{i=1}^n s_iw_i \in V(A^*)$ and $x \in X$ we get an element $\varphi(x) \in V(X)$. In this case we can ask whether $\varphi(x) = \psi(x)$ holds or not for some given $\varphi, \psi \in V(A^*)$ and $x \in X$. A pair $(\varphi, \psi) \in V(A^*)$ will be called a *linear equation*, and note that this is now a more general kind of equation since $A^* \subseteq V(A^*)$. We write $(X, \alpha) \models (\varphi, \psi)$; (X, α) satisfies the equation (φ, ψ) , if for every $x \in X$, $\varphi(x) = \psi(x)$.

Example 14. Let (X, α) be the weighted automata on $A = \{a, b, c\}$ with weights on \mathbb{Z}_3 given by the following diagram:



Then one easily verifies the following linear equations are satisfied by (X, α) : $ac = c$, $a + c = b$, and $2c + b = b^2$.

We have that $\mathbf{free}_w(X, \alpha)$ is the following automaton:



Hence, $\mathbf{Eq}_w(X, \alpha)$ is the congruence generated by

$$\{c^2 = c^3, b^3 = \epsilon, a^3 = \epsilon, ab = \epsilon, ba = \epsilon, ac = c, bc = c, ca = c, cb = c\}.$$

Observe that none of the linear equations $a + c = b$ nor $2c + b = b^2$ can be deduced from $\mathbf{Eq}_w(X, \alpha)$, even though (X, α) satisfies both of them. □

As the previous example shows, equations of the form $(\varphi, \psi) \in V(A^*) \times V(A^*)$ that are satisfied by (X, α) are interesting. We now turn to define the notion of linear automata.

For a field \mathbb{K} , let $\text{Vec}_{\mathbb{K}}$ denote the category of vector spaces over \mathbb{K} with linear maps as morphisms. A \mathbb{K} -linear automaton, or $\text{Vec}_{\mathbb{K}}$ automaton, on an alphabet A is a pair (S, α) , where S is an vector space and $\alpha : S \rightarrow S^A$ is a linear map. As in weighted automata, we can have an initial state $s_0 \in \text{Vec}_{\mathbb{K}}(\mathbb{K}, S)$ and/or a colouring $c \in \text{Vec}_{\mathbb{K}}(S, \mathbb{K})$. Notice that s_0 is completely determined by its value at 1, so we can identify s_0 with the element $s_0 := s_0(1) \in S$. Observe that every weighted automaton over a field \mathbb{K} yields a $\text{Vec}_{\mathbb{K}}$ automaton and conversely. A pointed $\text{Vec}_{\mathbb{K}}$ automaton (S, s_0, α) is a H -algebra for the endofunctor $H : \text{Vec}_{\mathbb{K}} \rightarrow \text{Vec}_{\mathbb{K}}$ given by $H(S) = \mathbb{K} + A \times S$ (formally, $H(S)$ is the sum of the space \mathbb{K} and A copies of the space S), and a coloured $\text{Vec}_{\mathbb{K}}$ automaton (S, c, α) is an I -coalgebra for the endofunctor $I : \text{Vec}_{\mathbb{K}} \rightarrow \text{Vec}_{\mathbb{K}}$ given by $I(S) = \mathbb{K} \times S^A$.

The initial H -algebra is $(V(A^*), \epsilon, \tau)$ where τ is given by $\tau(\sum s_i w_i)(a) = \sum s_i(w_i a)$, and for any pointed $\text{Vec}_{\mathbb{K}}$ automaton (S, s_0, α) the unique H -algebra morphism $r_{s_0} : V(A^*) \rightarrow S$ is given by $r_{s_0}(\sum s_i w_i) = \sum s_i w_i(s_0)$. Dually, the final I -coalgebra is $(\mathbb{K}^{A^*}, \hat{\epsilon}, \hat{\tau})$, where $\hat{\epsilon}(f) = f(\epsilon)$ and $\hat{\tau}(f)(a) = f_a$, and for any coloured $\text{Vec}_{\mathbb{K}}$ automaton (S, c, α) the unique I -coalgebra morphism $o_c : S \rightarrow \mathbb{K}^{A^*}$ is given by $o_c(s)(w) = c(w(s))$.

A $\text{Vec}_{\mathbb{K}}$ automaton (S, α) satisfies the linear equation $(\phi, \psi) \in V(A^*) \times V(A^*)$, denoted by $(S, \alpha) \models (\phi, \psi)$, if for any $s \in S$ we have that $\phi(s) = \psi(s)$.

An equivalence relation C on $V(A^*)$ is a linear congruence on $V(A^*)$ if:

- (i) $(\varphi_1, \psi_1), (\varphi_2, \psi_2) \in C$ imply $(\varphi_1 + \varphi_2, \psi_1 + \psi_2) \in C$.
- (ii) $(\varphi, \psi) \in C, k \in \mathbb{K},$ and $w \in A^*$ imply $(k\varphi, k\psi), (w\varphi, w\psi), (\varphi w, \psi w) \in C$.

Where, for $\phi = \sum_{i=1}^n k_i w_i \in V(A^*)$, $k\phi, w\phi,$ and ϕw are defined as

$$k\phi := \sum_{i=1}^n k k_i w_i, \quad w\phi = \sum_{i=1}^n k_i w w_i, \quad \phi w = \sum_{i=1}^n k_i w_i w.$$

Observe that if C is a linear congruence on $V(A^*)$ then the set $V(A^*)/C$ has the structure of a $\text{Vec}_{\mathbb{K}}$ pointed automaton.

Similarly to the case of weighted automata, we will define $\mathbf{free}_l(S, \alpha)$ and $\mathbf{Eq}_l(S, \alpha)$ by taking the following steps:

1. Define the pointed $\text{Vec}_{\mathbb{K}}$ automaton $\prod(S, \alpha) = (\prod_{s \in S} S, \Delta, \hat{\alpha})$ where $\hat{\alpha}$ is the product of $\alpha, |S|$ times, that is $\hat{\alpha}(x)(a)(s) = \alpha(x(s))(a)$, and $\Delta \in \prod_{s \in S} S$ is given by $\Delta(s) = s$. Then, by initiality of $(V(A^*), \epsilon, \tau)$, we get a unique H -algebra morphism $r_{\Delta} : V(A^*) \rightarrow \prod(S, \alpha)$.
2. Define $\mathbf{free}_l(S, \alpha)$ and $\mathbf{Eq}_l(S, \alpha)$ as

$$\mathbf{free}_l(S, \alpha) := V(A^*) / \ker(r_{\Delta}), \text{ and } \mathbf{Eq}_l(S, \alpha) := \ker(r_{\Delta}).$$

Notice that $\mathbf{Eq}_l(S, \alpha)$ is a linear congruence on $V(A^*)$.

Theorem 15. $\mathbf{Eq}_l(S, \alpha)$ is the maximum set of equations satisfied by (S, α) .

A set of coequations is a subspace $D \subseteq \mathbb{K}^{A^*}$. We define $(S, \alpha) \models D$, and say (S, α) satisfies the set of coequations D , as follows:

$$(S, \alpha) \models D \Leftrightarrow \forall c \in \text{Vec}_{\mathbb{K}}(S, \mathbb{K}), s \in S \ o_c(s) \in D.$$

For a family $\{S_i\}_{i \in I}$ of vector spaces, we denote the coproduct of that family by $\coprod_{i \in I} S_i$. The minimum set of coequations satisfied by (S, α) is obtained by the following construction:

1. Define the coloured $\text{Vec}_{\mathbb{K}}$ automaton $\coprod(S, \alpha) = (\coprod_{c \in \text{Vec}_{\mathbb{K}}(S, \mathbb{K})} S, \tilde{c}, \tilde{\alpha})$ where $\tilde{\alpha}$ is the coproduct of the morphism $\alpha, |\text{Vec}_{\mathbb{K}}(S, \mathbb{K})|$ times in $\text{Vec}_{\mathbb{K}}$, that is $\tilde{\alpha}(\sum(c_i, s_i))(a) = \sum(c_i, a(s_i))$, and \tilde{c} is the coproduct of all $c \in \text{Vec}_{\mathbb{K}}(S, \mathbb{K})$ which is given by $\tilde{c}(\sum(c_i, s_i)) = \sum c_i(s_i)$. Then, by finality of $(\mathbb{K}^{A^*}, \hat{e}, \hat{\tau})$, we get a unique I -coalgebra morphism $o_{\tilde{c}} : \coprod(S, \alpha) \rightarrow \mathbb{K}^{A^*}$.
2. Define $\mathbf{cofree}_I(S, \alpha)$ and $\mathbf{coEq}_I(S, \alpha)$ as

$$\mathbf{cofree}_I(S, \alpha) = \mathbf{coEq}_I(S, \alpha) = \text{Im}(o_{\tilde{c}}).$$

Theorem 16. $\mathbf{cofree}_I(S, \alpha)$ is the minimum set of coequations satisfied by (S, α) .

From the previous constructions we get the following result.

Theorem 17. For a linear congruence C , $\mathbf{free}_I \circ \mathbf{cofree}_I(V(A^*)/C) = V(A^*)/C$.

The inclusion from right to left above does not hold in general when considering semimodules (over a semiring) instead of vector spaces. Let \mathbb{N} be the semiring of natural numbers with the usual sum and product, $A = \{a, b\}$, and let C be the linear congruence on $V(A^*)$ associated to the partition $\{\{0\}, V(A^*) \setminus \{0\}\}$ of $V(A^*)$. Then $V(A^*)/C \cong \mathbb{B}$, the Boolean semiring, where the action of \mathbb{N} on \mathbb{B} is given by $n \cdot x = 0$ if and only if $n = 0$ or $x = 0$. However, one can verify that $\mathbf{cofree}_I(V(A^*)/C)$ has only one element and therefore it satisfies any identity. It follows that $V(A^*)/C$ cannot be a subset of $\mathbf{free}_I \circ \mathbf{cofree}_I(V(A^*)/C) = 1$.

Similarly to the case of weighted automata we can show that for a $\text{Vec}_{\mathbb{K}}$ automaton (S, α) and a linear congruence C we have that

$$(S, \alpha) \models C \text{ if and only if } (S, \alpha) \models \mathbf{coEq}_I(V(A^*)/C).$$

Example 18. Let $A = \{x, y\}$, and let $C = \langle xy = yx \rangle$ be the linear congruence generated by the equation $xy = yx$. Then the $\text{Vec}_{\mathbb{K}}$ automaton $V(A^*)/C$ is isomorphic to $\mathbb{K}[x, y]$, the ring of polynomials on indeterminates x and y with coefficients in \mathbb{K} . Here the transition function on $\mathbb{K}[x, y]$ is (right) multiplication, that is, for a polynomial $p(x, y) \in \mathbb{K}[x, y]$ we have that $x(p(x, y)) = p(x, y)x$, and $y(p(x, y)) = p(x, y)y$. Then, $\mathbf{cofree}_I(V(A^*)/C)$ is the set

$$\left\{ f \in \mathbb{K}^{A^*} \mid \forall w_1, w_2 \left(|w_1|_x = |w_2|_x \wedge |w_1|_y = |w_2|_y \Rightarrow f(w_1) = f(w_2) \right) \right\},$$

where $|w|_x$ is the number of x 's in the word $w \in A^*$. We can go a little bit further and notice that $\mathbf{cofree}_I(V(A^*)/C) \cong \mathbb{K}^{\mathcal{M}(\mathbb{K}[x, y])}$ where $\mathcal{M}(\mathbb{K}[x, y])$ are the monic monomials in $\mathbb{K}[x, y]$. □

Example 19. Let $A = \{x, y\}$, and, for a fixed $k \in \mathbb{K}$, let $C = \langle xy = yx, y - k = 0 \rangle$ be the linear congruence generated by the equations $xy = yx$ and $y - k = 0$. Then the $\text{Vec}_{\mathbb{K}}$ automaton $V(A^*)/C$ is isomorphic to $\mathbb{K}[x, y]/\langle y - k \rangle \cong \mathbb{K}[x]$, where $\langle y - k \rangle$ is the ideal generated by $y - k$. A similar calculation as in the previous example shows that $\mathbf{cofree}_l(V(A^*)/C) \cong \mathbb{K}\{x\}^* \cong \mathbb{K}^{\mathbb{N}}$. \square

Example 20. Let A be a finite alphabet, and C a linear congruence on $V(A^*)$ such that for any $x, y \in A$, $(xy, yx) \in C$. We claim that $C = \langle C_0 \rangle$ for some finite $C_0 \subseteq C$. In fact, if $A = \{x_1, \dots, x_n\}$ then $V(A^*)/C \cong \mathbb{K}[x_1, \dots, x_n]/I$ for the ideal I of $\mathbb{K}[x_1, \dots, x_n]$ given by $I = \{\phi - \psi \mid (\phi, \psi) \in C\}$, which is an ideal of $\mathbb{K}[x_1, \dots, x_n]$ since C is a linear congruence (here $\phi - \psi$ is calculated as in $\mathbb{K}[x_1, \dots, x_n]$). Then by Hilbert basis theorem we have that I is finitely generated, say $I = \langle \varphi_1, \dots, \varphi_m \rangle$. It follows that

$$C = \langle \{x_i x_j = x_j x_i \mid 1 \leq i < j \leq n\} \cup \{\varphi_l = 0 \mid 1 \leq l \leq m\} \rangle.$$

\square

Proposition 21. *Let $S \subseteq \mathbb{K}^{A^*}$ be a set that is closed under left derivatives. Then $(S, \alpha) \subseteq \mathbf{cofree}_l \circ \mathbf{free}_l(S, \alpha)$.*

To complete a duality result in this case we still need to characterize systems S of the form $S = \mathbf{cofree}_l(V(A^*)/C)$ for some linear congruence C . The previous proposition is a first step, but a full duality result is an open problem.

6 Conclusion

All the results we proved concerning weighted automata can be easily adapted to the case of probabilistic automata, that is, we can replace $V(X)$ by $\mathcal{D}(X)$, where $\mathcal{D}(X) = \{f : X \rightarrow [0, 1] \mid \text{supp}(f) \text{ is finite and } \sum_{x \in \text{supp}(f)} f(x) = 1\}$, and replace the semiring \mathbb{S} by the real interval $[0, 1]$ to get similar results. In fact, the linear extension $\bar{\alpha} : V(X) \rightarrow V(X)^A$ of a function $\alpha : X \rightarrow V(X)^A$ is well-defined if we replace $V(X)$ by $\mathcal{D}(X)$ and the elements 0 and 1 in \mathbb{S} are the elements 0 and 1 in $[0, 1]$, respectively. In the future we plan to study other types of automata such as tree automata. Furthermore, we also want to understand the duality for linear weighted automata.

The closest related work is [3] which can be considered as a special case of our study here. While we allow automata with infinite sets of states, other dualities have concentrated on the finitary case [1, 8, 9, 12]. Another difference with those approaches is that we present a local duality, in which the alphabet is (possibly infinite but) fixed, while [8, 9, 12] consider all finite alphabets at the same time.

References

1. Adámek, J., Milius, S., Myers, R.S.R., Urbat, H.: Generalized eilenberg theorem i: local varieties of languages. In: Muscholl, A. (ed.) FOSSACS 2014 (ETAPS). LNCS, vol. 8412, pp. 366–380. Springer, Heidelberg (2014)

2. Arbib, M.A., Manes, E.G.: Foundations of system theory: the hankel matrix. *J. Comput. Syst. Sci.* **20**(3), 330–378 (1980)
3. Ballester-Bolinches, A., Cosme-López, E., Rutten, J.J.M.M.: The dual equivalence of equations and coequations for automata. CWI Technical report FM-1403, pp. 1–41 (2014, To appear in *Information and Computation*)
4. Bezhanishvili, N., Kupke, C., Panangaden, P.: Minimization via duality. In: Ong, L., de Queiroz, R. (eds.) *WoLLIC 2012*. LNCS, vol. 7456, pp. 191–205. Springer, Heidelberg (2012)
5. Bonchi, F., Bonsangue, M., Boreale, M., Rutten, J., Silva, A.: A coalgebraic perspective on linear weighted automata. *Inf. Comp.* **211**, 77–105 (2012)
6. Bonchi, F., Bonsangue, M.M., Hansen, H.H., Panangaden, P., Rutten, J., Silva, A.: Algebra-coalgebra duality in brzozowski’s minimization algorithm. *ACM Trans. Comput. Logic* **15**(1), 3 (2014)
7. Burri, S.N., Sankappanava, H.P.: A course in universal algebra. *Graduate Texts in Mathematic*, vol. 78. Springer, New York (1981)
8. Eilenberg, S.: *Automata, languages, and machines*, vol. B. Academy Press, New York (1976)
9. Gehrke, M., Grigorieff, S., Pin, J.É.: Duality and equational theory of regular languages. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008, Part II*. LNCS, vol. 5126, pp. 246–257. Springer, Heidelberg (2008)
10. Hungerford, T.W.: *Algebra*. *Graduate Texts in Mathematics*, vol. 73. Springer, New York (1974)
11. Kuich, W.: Semirings and formal power series. In: Rozenberg, G., Salomaa, A. (eds.) *Handbook of Formal Languages, Word, Language, Grammar*, vol. 1, pp. 609–677. Springer, Heidelberg (1997)
12. Petković, T.: Varieties of fuzzy languages. In: *Proceedings of International Conference on Algebraic Informatics*. Aristotle University of Thessaloniki (2005)
13. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. *Theor. Comput. Sci.* **249**(1), 3–80 (2000)
14. Schützenberger, M.P.: On the definition of a family of automata. *Inf. Control* **4**, 245–270 (1961)

Author Index

- Abbes, Samy I-63
Abed, Fidaa II-1
Aceto, L. I-76
Allender, Eric II-14
Arvind, V. II-26, II-38
Avni, Guy I-89
- Baartse, Martijn II-50
Babenko, Maxim II-62
Baier, Christel I-344
Barcelo, Neal II-75, II-90
Baswana, Surender II-102
Belmonte, Rémy II-115
Berlinkov, Mikhail I-103
Berthé, Valérie I-116
Boldi, Paolo I-3
Bonsangue, Marcello I-444
Bottesch, Ralph II-127
Boudou, Joseph I-129
Brandstädt, Andreas II-139
Braverman, Vladimir II-151
Bulatov, Andrei A. II-175
- Cadilhac, Michaël I-141
Caferov, Cafer II-187
Caragiannis, Ioannis II-1
Case, Adam II-199
Cesaratto, Eda I-116
Chen, Lin II-211
Chen, Ruiwen II-223
Choudhary, Keerti II-102
Chumbalov, Daniyar II-235
Coan, Brian II-274
Cord-Landwehr, Andreas II-248
Crépeau, Claude II-261
Crespi Reghizzi, Stefano I-154
- Dabrowski, Konrad K. II-139
Daviaud, Laure I-167
Della Monica, D. I-76
Di Crescenzo, Giovanni II-274
Dima, Cătălin I-179
- Droste, Manfred I-192
Dubslaff, Clemens I-344
Dück, Stefan I-192
Dumitran, Marius I-205
Durand, Bruno I-218
- Ésik, Zoltán I-18
Etscheid, Michael II-287
- Fábregas, I. I-76
Fafianie, Stefan II-299
Farhat, Saman II-483
Ferraioli, Diodato II-311
Fomin, Fedor V. II-115
Fournier, Hervé II-324
- Gajardo, Anahí I-231
Gajarský, Jakub II-336
Gál, Anna II-14
Ganian, Robert II-348
García-Marco, Ignacio II-361
Gavinsky, Dmitry II-127
Gawrychowski, Paweł I-243
Goldberg, Andrew V. II-62
Golovach, Petr A. II-115
Grellois, Charles I-256
Grilo, Alex Bredariol II-163
- Hagerup, Torben II-372
Hahn, Michael II-384
Hannula, Miika I-269
Harteringer, Tatiana R. II-395
Hella, Lauri I-281
Hoffmann, Udo II-407
Huang, Shenwei II-139
- Iacobelli, Giulio I-293
Ingólfssdóttir, A. I-76
- Jakobsen, Sune K. II-420
Joglekar, Pushkar II-38
Johnson, Matthew II-395

- Kaminski, Benjamin Lucien I-307
 Kaplan, Haim II-62
 Katoen, Joost-Pieter I-307
 Kaya, Barış II-187
 Kazmi, Raza Ali II-261
 Kerenidis, Iordanis II-163
 Kiefer, Sandra I-319
 Kim, Eun Jung II-348
 Kirsch, Jonathan II-274
 Kisielewicz, Andrzej I-331
 Klauck, Hartmut II-127
 Kleist, Linda II-407
 Kling, Peter II-75
 Köbler, Johannes II-26
 Koiran, Pascal II-361
 König, Daniel II-445
 Kontinen, Juha I-269
 Kosolobov, Dmitry II-432
 Krähhmann, Daniel I-344
 Kratsch, Stefan II-287, II-299
 Krebs, Andreas I-141, II-384
 Kupferman, Orna I-89
 Kutrib, Martin I-38
 Kuusisto, Antti I-281
- Lagerkvist, Victor I-357
 Lampis, Michael II-336
 Lange, Klaus-Jörn II-384
 Latte, Markus I-369
 Lenzner, Pascal II-248
 Limaye, Nutan II-324
 Liu, Zaoxing II-151
 Lohrey, Markus II-445
 Lonati, Violetta I-154
 Ludwig, Michael I-141, II-384
 Lutz, Jack H. II-199
- Mahajan, Meena II-324
 Mairesse, Jean I-63
 Makino, Kazuhisa II-336
 Mandal, Debasis II-459
 Mandrioli, Dino I-154
 Manea, Florin I-205
 Mansour, Yishay I-53
 Maubert, Bastien I-179
 Mauro, Jacopo I-382
 Mazowiecki, Filip I-394
 McGregor, Andrew II-472
 Meer, Klaus II-50
- Megow, Nicole II-211
 Meier, Arne I-281
 Melliès, Paul-André I-256
 Mertz, Ian II-14
 Milanič, Martin II-395
 Miltzow, Tillmann II-407
 Mitsou, Valia II-336
 Mneimneh, Saad II-483
 Mnich, Matthias II-287
 Montanari, Sandro II-493
 Mühenthaler, Moritz II-505
- Niewerth, Matthias I-369
 Nugent, Michael II-75, II-90
- O'Donnell, Ryan II-187
 Ochremiak, Joanna I-394
 Ollinger, Nicolas I-231
 Ordyniak, Sebastian II-336
- Paperman, Charles I-141, I-167
 Paulusma, Daniël II-139, II-395
 Pauly, Arno I-407
 Pavan, A. II-459
 Penna, Paolo II-493
 Perrot, Kévin I-419
 Philip, Geevarghese II-517
 Pinchinat, Sophie I-179
 Pradella, Matteo I-154
 Pruhs, Kirk II-75, II-90
- Rahn, Mona II-529
 Rai, Ashutosh II-517
 Ramanujan, M.S. II-115
 Rattan, Gaurav II-26, II-38
 Regnault, Damien I-432
 Rémila, Éric I-419, I-432
 Rischke, Roman II-211
 Röglin, Heiko II-287
 Romashchenko, Andrei I-218, II-235
 Rotondo, Pablo I-116
 Rutten, Jan I-444
- Salamanca, Julian I-444
 Saurabh, Saket II-517
 Savchenko, Ruslan II-62
 Say, A.C. Cem II-187
 Schäfer, Guido II-529
 Schmid, Markus L. II-542

- Schnoor, Henning II-555
Schubert, Jana I-344
Schweitzer, Pascal I-319
Scquizzato, Michele II-75, II-90
Selman, Erkal I-319
Sikora, Jamie II-163
Singh, Tejasvam II-151
Sitters, René II-567
Skvortsov, Evgeny S. II-175
Srinivasan, Srikanth II-324
Stougie, Leen II-211
Straszak, Damian I-243
Szeider, Stefan II-348
Szykuła, Marek I-103, I-331
- Tavenas, Sébastien II-361
Tench, David II-472
Torres-Avilés, Rodrigo I-231
Tribastone, Mirco I-293
- Vallée, Brigitte I-116
van Ee, Martijn II-567
Vandin, Andrea I-293
Ventre, Carmine II-311
- Verbitsky, Oleg II-26
Verschae, José II-211
Vinodchandran, N.V. II-151, II-459
Viola, Alfredo I-116
Virtema, Jonni I-269
Vollmer, Heribert I-269, I-281
Vorotnikova, Sofya II-472
Voudouris, Alexandros A. II-1
Vu, Hoa T. II-472
- Weller, Mathias II-62
Witkowski, Adam I-394
Woizekowski, Oliver II-555
Wu, Xiaowei II-577
- Yang, Lin F. II-151
Yu, Bin II-601
- Zavattaro, Gianluigi I-382
Zehavi, Meirav II-589
Zhang, Chenzi II-577
Zhang, Miaomiao II-601
Zu, Quan II-601