Computer Science Logic 2011

25th International Workshop
20th Annual Conference of the EACSL
CSL'11, September 12–15, 2011, Bergen, Norway

Edited by

Marc Bezem

Editor

Marc Bezem
Department of Informatics
University of Bergen
bezem@ii.uib.no

ACM Classification 1998
A.0 Conference Proceedings; D. Software; F. Theory of Computation;
G.2.2 Graph Theory; G.3 Probability and Statistics; I.2 Artificial Intelligence

ISBN 978-3-939897-32-3

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-32-3.

Publication date
September, 2011

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

Digital Object Identifier: 10.4230/LIPIcs.CSL.2011.i

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

www.dagstuhl.de/lipics

Contents

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Retiring President's Address

Editors's Preface

The annual conference of the European Association for Computer Science Logic (EACSL), CSL 2011, was held in Bergen, Norway, from 12 to 15 September 2011. CSL started as a series of international workshops on Computer Science Logic, and then at its sixth meeting became the Annual Conference of the EACSL. This conference was the 25th workshop and 20th EACSL conference; it was organized by the Department of Informatics of the University of Bergen.

CSL 2011 was preceded by TYPES 2011, the 18th Workshop Types for Proofs and Programs (8–11 September), and by the pre-conference workshop Epsilon Calculus and Constructivity organized by Matthias Baaz and Georg Moser (11 September). The technical program of TYPES 2011 was independent and the TYPES 2011 post-proceedings will be published elsewhere.

The Ackermann Award is the EACSL Outstanding Dissertation Award for Logic in Computer Science. The recipient of the Ackermann Award for 2011 is Benjamin Rossman, and the award was officially presented at the conference (14 September). The citation of the award, an abstract of the thesis, and a biographical sketch of the recipient may be found on page xv of the proceedings.

This is the first year that the CSL proceedings are not published as a Springer LNCS but in the series LIPIcs. In response to the call for papers, a total of 116 abstracts were registered and 91 of these were followed by full papers submitted to CSL 2010. The Program Committee (PC) selected 37 papers for presentation at the conference and publication in these proceedings. Each paper was assigned to four PC members. In the call for papers, authors were encouraged to include a well written introduction. One of the four PC members for each paper had the particular task to assess the accessibility of the introduction to the computer science logic community at large. Also this year the overall high quality of the submissions made that many good papers had to be rejected due to lack of space. In addition to the contributed talks, CSL 2011 had four invited speakers: Thomas Ehrhard (Université Paris Diderot), Martin Otto (Technische Universität Darmstadt), Moshe Vardi (Rice University), Frank Wolter (University of Liverpool). Abstracts of the invited talks are included in the proceedings.

I thank the PC and all external reviewers for their help in reviewing the papers. I also thank the Organizing Committee, in particular Michal Wałicky and Isolde Adler, for their capable efforts in organizing the conference. The conference received generous support from the Research Council of Norway and from the Kurt Gödel Society. We are grateful to these institutions for their sponsorship. Special thanks go to Janos Makowsky, our President during an extended term 2004–2010. The EACSL has flourished under his presidency, thanks to his many good initiatives. Among these, I just mention the creation of the Ackermann Award and his consistent efforts to move with the CSL proceedings from a commercial publisher to open access. A detailed personal account can be found in his Retiring President's Address at the end of the proceedings.

September 2011 Marc Bezem

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Conference Organization

Program Committee

Samson Abramsky (Oxford)
Andrea Asperti (Bologna)
Franz Baader (Dresden)
Matthias Baaz (Vienna)
Johan van Benthem (Amsterdam/Stanford)
Marc Bezem (Bergen, chair)
Patrick Blackburn (Nancy)
Andreas Blass (Michigan)
Jan van den Bussche (Hasselt)
Thierry Coquand (Gothenburg)
Nachum Dershowitz (Tel Aviv)
Valentin Goranko (Copenhagen)
Erich Grädel (Aachen)
Wiebe van der Hoek (Liverpool)
Bart Jacobs (Nijmegen)
Reinhard Kahle (Lisbon)
Stephan Kreutzer (Oxford)
Viktor Kuncak (Lausanne)
Daniel Leivant (Indiana)
Benedikt Löwe (Amsterdam)
Jean-Yves Marion (Nancy)
Eugenio Moggi (Genova)
Albert Rubio (Barcelona)
Anton Setzer (Swansea)
Alex Simpson (Edinburgh)
John Tucker (Swansea)
Paweł Urzyczyn (Warsaw)
Helmut Veith (Vienna)
Andrei Voronkov (Manchester)

Organizing Committee

Isolde Adler (Frankfurt)
Marc Bezem (Bergen)
Magne Haveraaen (Bergen)
Michał Walicki (Bergen)

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem

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

External Reviewers

Andreas Abel
Luca Aceto
Jesse Alama
Thorsten Altenkirch
Robert Atkey
Jeremy Avigad
Philippe Balbiani
Pablo Barceló
Bruno Barras
Peter Baumgartner
Josh Berdine
Małgorzata Biernacka
Jasmin Christian Blanchette
Udi Boker
Filippo Bonchi
Guillaume Bonfante
Christopher Broadbent
Sabine Broda
Paola Bruscoli
Véronique Bruyère
Harry Buhrman
Martin Bunder
Sascha Böhme
Olivier Carton
Arthur Charguéraud
Jacek Chrząszcz
Denis Cousineau
Nadia Creignou
Pierre-Evariste Dagand
Ugo Dal Lago
Victor Dalmau
Nils Anders Danielsson
Norman Danner
Anuj Dawar
David De Frutos-Escrig
Hans De Nivelle
Giorgio Delzanno
Mariangiola Dezani
Pietro Di Gianantonio
Clare Dixon
Jacques Duparc
Arnaud Durand
Rüdiger Ehlers
Fabian Emmes
Jörg Endrullis
Martin Escardo
Kousha Etessami
Claudia Faggian
Oliver Fasching
Wan Fokkink
Murdoch Gabbay
John Gallagher
Silvia Ghilezan
Mayer Goldberg
Rajeev Gore
Benjamin Gregoire
Stefano Guerrini
Ferruccio Guidi
Makoto Hamana
Peter Hancock
Helle Hvid Hansen
Ichiro Hasuo
Chris Heunen
Roger Hindley
Dieter Hofbauer
Douglas Howe
Ullrich Hustadt
Dmitry Itsykson
Swen Jacobs
Barbara Jobstmann
Jean-Pierre Jouannaud
Stefan Kahrs
Delia Kesner
Ulrich Kohlenbach
Yoshiharu Kojima
Leszek Kołodziejczyk
Boris Konev
Igor Konnov
Roman Kontchakov
Laura Kovács
Oliver Kullman
Oliver Kutz
Timo Kötzing
Raul Leal
Stephane Lengrand
Pierre Lescanne
Daniel R. Licata
Alexei Lisitsa
Liquori Luigi
Christof Löding

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Jorik Mandemaker
Andrea Masini
Fabio Massacci
Ralph Matthes
Aart Middeldorp
Michał Moskal
Larry Moss
Andrzej Murawski
Rasmus Ejlers Møgelberg
Iman Narasamdya
Sara Negri
Frank Neven
Linh Anh Nguyen
Vivek Nigam
Isabel Oitavem
Paulo Oliva
Martin Otto
Prakash Panangaden
Luca Paolini
Michel Parigot
Pawel Parys
Dirk Pattinson
Rasmus Lerchedahl Petersen
Rafael Peñaloza
Adolfo Piperno
Alberto Policriti
Randy Pollack
Andrew Polonsky
Alexander Rabinovich
George Rahonis
Jakob Rehof
Wilmer Ricciotti
Eike Ritter
Enric Rodríguez-Carbonell
Luca Roversi
Paul Rozière
Ji Ruan
Jan Rutten
Stefan Rümmele
Joshua Sack
Mehrnoosh Sadrzadeh
Antonino Salibra
Ulrike Sattler
Thomas Scanlon
Sven Schewe
Manfred Schmidt-Schauss
Aleksy Schubert
Carsten Schürmann
Thomas Schwentick
Philip Scott
Luciano Serafini
Traian Serbanuta
Olivier Serre
Gert Smolka
Bas Spitters
Colin Stirling
Thomas Strahm
Tomoyuki Suzuki
Wouter Swierstra
Balder Ten Cate
Lidia Tendera
Hayo Thielecke
Hans Tompits
Sergei Tupailo
Nikos Tzevelekos
Tarmo Uustalu
Jouko Vaananen
Steffen Van Bakel
Benno Van Den Berg
Rob Van Glabbeek
Vincent Van Oostrom
Lionel Vaux
Varmo Vene
Jamie Vicary
Heribert Vollmer
Yanjing Wang
Daniel Weller
Ronny Wichers Schreur
Frank Wolter
Michał Wrona
Michael Zakharyaschev
Ana Zamanski
Noam Zeilberger
Martin Zimmermann
Jeffery Zucker
Florian Zuleger

The Ackermann Award 2011

Report of the Jury

The seventh Ackermann Award will be presented at this CSL'11, held in Bergen, Norway. This is the fifth year the EACSL Ackermann Award is generously sponsored. Our sponsor for the period of 2011-2013 is the Kurt Gödel Society (KGS). Besides providing financial support for the Ackermann Award, the KGS also committed itself to inviting the receiver of the Award for a special lecture to be given in Vienna.

Eligible for the 2011 Ackermann Award were PhD dissertations in topics specified by the EACSL and LICS conferences, which were formally accepted as Ph.D. theses at a university or an equivalent institution between 1.1. 2009 and 31.12. 2010. The Jury received 10 nominations for the Ackermann Award 2011. The candidates came from 9 different countries in Europe, North America, South America and Australia, and received their degrees in 6 different countries in Europe and North America.

The topics covered the full range of Logic and Computer Science as represented by the LICS and CSL Conferences. All the submissions were of very high standard and contained outstanding results in their particular domain. The Jury wishes to congratulate all the nominated candidates for their outstanding work. The Jury encourages them to continue their scientific careers, and hopes to see more of their work in the future. The Jury decided unanimously to give the **Ackermann Award 2011** to

Benjamin Rossman.

Citation

Benjamin Rossman receives the Ackermann Award 2011 of the European Association of Computer Science Logic (EACSL) for his thesis

Average Case Complexity of Detecting Cliques.

The thesis represents a breakthrough in our understanding of circuit complexity. It settles a long-standing open question on the expressive power of first-order logic on ordered graphs and does so by developing innovative methods of proving lower bounds on the complexity of circuits. These methods advance the state of the art and represent the most significant breakthrough in circuit complexity in many years.

Background

While the main results in the thesis are in the area of circuit complexity, the motivation for the work comes from questions of logic and, ultimately, the results obtained have a strong connection with these motivating questions. Indeed, they also provide one of the most significant breakthroughs in the field of finite model theory in many years.

The motivating problem in logic is the following. Can we express more with first-order logic using $k+1$ variables than we can with k on ordered finite graphs? This question is deceptively simple to state, but turns out to be very difficult to answer. If we drop either of the restrictions to finite or to ordered graphs, it is easy to show that an infinite hierarchy of expressive power is obtained by increasing the number of variables. On the other hand, if, instead of graphs, we consider linear orders with unary relations only, it is known that every

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

first-order sentence is equivalent to one with just 3 variables. The question for finite linear orders with one binary relation, i.e. finite ordered graphs, however, turns out to be tied to difficult complexity theoretic considerations.

The connections with complexity theory emerge from the work of Neil Immerman on descriptive complexity in the 1980s. Following Fagin's proof that the class of problems definable in existential second-order logic is exactly the complexity class NP, Immerman established the connection between a number of different complexity classes and corresponding definability classes. Most of these correspondences require us to assume that structures are ordered or even that they have rich arithmetic predicates available as these are necessary in order to give defining formulas the power to simulate computations. Indeed, the weaker the complexity class, the richer the fragment of arithmetic required. Thus, while NP is captured by existential second-order logic without any requirement for order, the characterization of P as definability in least fixed-point logic only works on ordered structures and AC_0 is captured by first-order logic with order *and* arithmetic predicates.

While the work on descriptive complexity had raised hopes that model-theoretic methods could be deployed to prove complexity lower bounds, the best known such methods really only provided inexpressibility results in the absence of order. Thus, the challenge before the field of finite model theory was to develop methods that could be used to establish lower bounds on *ordered* structures. An iconic problem, representing this challenge, was the question of showing that increasing the number of variables leads to an increase in expressive power on ordered finite graphs.

There is a first-order sentence with k variables that expresses that a graph contains a clique on k vertices. This sentence does not require an order. But are k variables really necessary? Or, at least, can one show that no fixed number of variables suffice to express the clique problem for all k? The question was posed in essentially this form by Immerman in 1982. It was beyond the boundary of the model-theoretic methods available yet appeared simpler than a full-fledged complexity lower bound. Indeed, it is closely connected to a question of circuit complexity. A first-order sentence ϕ with k variables can be translated to a family of circuits C_n of bounded depth (bounded by the quantifier depth of the sentence) and size n^k, such that C_n accepts encodings of those graphs of size n that satisfy ϕ. Conversely, any such family translates into a first-order sentence, but one that requires an order and arithmetic relations in addition to the graph relation in its vocabulary. Thus, Immerman's question could be answered if one could show a suitable lower bound on the size of circuits required to solve the clique problem. That is, that there is no fixed k such that circuits of bounded depth and size n^k suffice to decide the l-clique problem for all l. This was widely believed, but considered beyond the methods of circuit complexity at the time.

This lower bound is what Rossman establishes. He proves that there is no family of constant-depth circuits of size $O(n^{k/4})$ that can decide the k-clique problem. It follows that there is no sentence of first-order logic using fewer than $k/4$ variables, even with order and *arbitrary* arithmetic predicates that can express the existence of a k-clique. He then shows (using a previously unpublished construction due to Immerman) that it follows that for each k there is a sentence with $k+1$ variables that is not equivalent to one with k.

In the 1980s, the clique problem was well-studied in the context of circuit lower bounds. Methods based on Håstad's switching lemma were used to establish a trade-off between the size and depth of circuits required. In particular, Beame showed in 1990 that any family of circuits of depth d that decided k-clique would require size $n^{\Omega(k/d^2)}$. However, it was widely held that the methods could not be extended to obtain a lower bound that was independent of depth and this is where Rossman has made a breakthrough.

Rossman's Contribution

The innovation in Rossman's method is to combine methods based on the switching lemma with considering sparse random graphs. Let $G(n,p)$ denote the probability distribution on graphs on the set of vertices $\{1,\ldots,n\}$ obtained by letting each pair (i,j) with $i<j$ have an edge with probability p. Then $p(n)=n^{-2/(k-1)}$ is the threshold for the existence of k-cliques. That is, with p much below this, the probability that a graph in $G(n,p)$ contains a k-clique goes to 0 while with p much higher than the threshold, it goes to 1. What Rossman proves is that for any family of constant depth circuits of size $O(n^{k/4})$, with high probability the same answer is obtained on a random graph in $G(n,p)$ as on one in which k-clique has been planted.

This result not only establishes a lower-bound on the worst-case complexity of the clique problem, it shows that the *average-case* complexity is worse than had been shown before. To be precise, it shows that any algorithm which can be represented as a bounded-depth family of circuits (i.e. it is sufficiently parallelizable) must take time $\Omega(n^{k/4})$ on average to decide the presence of a k-clique.

In another set of results, Rossman considers *monotone* circuits (i.e. circuits that do not use `not` gates). This is another class of circuits where lower bounds have previously been obtained. Razborov showed in 1985 that the k-clique problem cannot be decided by a family of monotone circuits of size $O((n/\log^2 n)^k)$ and the bound was subsequently further improved by Alon and Boppana. What Rossman establishes is new lower bounds on the average case complexity of monotone circuits for the clique problem. This is again done by considering sparse random graphs at the threshold for the existence of k-clique.

In some sense the results of this thesis close a line of research within finite model theory. The open problem which had inspired much interesting work has been settled. Moreover, the breakthrough has not come from extending methods from logic as had been hoped at some point but rather, it is a breakthrough in complexity theory that has settled the long-standing problem in logic. Yet, this breakthrough does provide new methods and which can and will be applied to other problems.

Among the methods that should be highlighted are a new notion of sensitivity which provides a powerful analytical tool for studying bounded-depth circuits that work on graph properties. It is this that enables Rossman to extend methods based on the switching lemma far beyond their previous use. There are also new combinatorial tools among which one should mention the quasi-sunflower lemma which provide an interesting extension of the Erdös-Renyi sunflower lemma in the "average case".

Finally, the thesis is to be commended for its presentational style, which makes difficult mathematical material so accessible.

Biographic Sketch

Benjamin Rossman received his B.A. and M.A. degrees in Mathematics from the University of Pennsylvania in 2001 and 2002 respectively. He completed his PhD in 2010 at the Massachusetts Institute of Technology under the supervision of Madhu Sudan. Since September 2010 he is at the Tokyo Institute of Technology supported by an NSF Mathematical Sciences Postdoctoral Research Fellowship. He has twice (in 2003 and 2005) received the Kleene award for best student paper at the IEEE Symposium on Logic in Computer Science. His thesis received the George M. Sprowls award at MIT.

The Jury

The Jury for the **Ackermann Award 2011** consisted of eight members, three of them ex officio, namely the president and the vice-president of EACSL, and one member of the LICS organizing committee.

The members of the Jury were

- A. Atserias (Barcelona, Spain),
- T. Coquand (Gothenburg, Sweden),
- P.-L. Curien (Paris, France),
- A. Dawar (Cambridge, U.K., Vice-president of EACSL),
- J.-P. Jouannaud (Paris, France and Beijing, China),
- D. Niwinski (Warsaw, Poland, President of EACSL),
- L. Ong (Oxford, U.K., LICS representative), and
- W. Thomas (Aachen, Germany).

They were helped by the non-voting coordinator of the Jury, J.A. Makowsky (Haifa, Israel, Secretary of the Jury and Member of the EACSL Board).

June 2011 Anuj Dawar, Johann A. Makowsky, and Damian Niwinski

The Ackermann Award

The EACSL Board decided in November 2004 to launch the EACSL Outstanding Dissertation Award for Logic in Computer Science, the **Ackermann Award**, The award is named after the eminent logician Wilhelm Ackermann (1896-1962), mostly known for the Ackermann function, a landmark contribution in early complexity theory and the study of the rate of growth of recursive functions, and for his coauthorship with D. Hilbert of the classic *Grundzüge der Theoretischen Logik*, first published in 1928. Translated early into several languages, this monograph was the most influential book in the formative years of mathematical logic. In fact, Gödel's completeness theorem proves the completeness of the system presented and proved sound by Hilbert and Ackermann. As one of the pioneers of logic, W. Ackermann left his mark in shaping logic and the theory of computation. Details concerning the Ackermann Award and a biographic sketch of W. Ackermann were published in the CSL'05 proceedings and can also be found at `http://www.eacsl.org/award.html`.

The Ackermann Award is presented to the recipients at the annual conference of the EACSL. The Jury is entitled to give more than one award per year. The award consists of a diploma, an invitation to present the thesis at the CSL conference, the publication of the abstract of the thesis and the citation in the CSL proceedings, and travel support to attend the conference.

Previous winners of the Ackermann Award

2005, Oxford:
Mikołaj Bojańczyk from Poland,
Konstantin Korovin from Russia, and
Nathan Segerlind from the USA.

2006, Szeged:
Balder ten Cate from The Netherlands, and
Stefan Milius from Germany.

2007, Lausanne:
Dietmar Berwanger from Germany and Romania,
Stéphane Lengrand from France, and
Ting Zhang from the People's Republic of China.

2008, Bertinoro:
Krishnendu Chatterjee from India.

2009, Coimbra:
Jakob Nordström from Sweden.

2010, Brno:
No award was given.

Detailed reports of the Jury, and the work of the recipients, appeared in the CSL'05, CSL'06, CSL'07, CSL'08, CSL'09 and CSL'10 proceedings, and are also available via the EACSL homepage `http://www.eacsl.org/award.html`.

Resource lambda-calculus: the differential viewpoint*

Thomas Ehrhard

CNRS, PPS, UMR 7126, Univ Paris Diderot, Sorbonne Paris Cité
F-75205 Paris, France
thomas.ehrhard@pps.jussieu.fr

Abstract

Milner's π calculus features a clear dichotomy between replicable and non-replicable resources, very much in the spirit of Linear Logic (LL). Analyzing Milner's encoding of the lazy λ-calculus in the π-calculus, Boudol introduced the λ-*calculus with resources* [1, 2] where functions can be applied to bags made of replicable and non-replicable arguments. This refinement of the syntax required to stick to a lazy reduction strategy implemented with explicit substitutions, used to postpone linear substitutions of non replicable resources.

Motivated by the discovery of denotational models of LL such as [3] where all morphisms of the associated cartesian closed category can be differentiated, we introduced in [4] the *differential* λ-*calculus* which features two ways of applying a function to an argument: the standard one and a linear one, implementing differentiation. This approach allows to generalize Boudol's idea: linear β-reduction – understood now as differentiation – can be performed everywhere in terms, and explicit substitutions are not needed anymore. Moreover, these differential operations fit very well in the LL framework: differentiation appears as a logical rule dual to dereliction, and the nice par/tensor symmetry of multiplicative LL extends to the exponentials, see [5]. In the conclusion of [6], Girard contemplated the possibility of introducing differential ideas in LL. There were probably very good reasons for not doing so at this early stage, since differential constructs are incompatible with both determinism and totality.

We present differential linear logic and its models, the associated resource and differential λ-calculi, and the Taylor expansion of promotion boxes. We also describe an *antiderivative* which seems to be available in many models of differential LL, and we present a very simple categorical axiom for this operation.

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.1

References

1 Gérard Boudol. The lambda-calculus with multiplicities. Technical Report 2025, INRIA Sophia Antipolis, 1993.
2 Gérard Boudol, Pierre-Louis Curien, and Carolina Lavatelli. A semantics for lambda calculi with resource. *Mathematical Structures in Computer Science*, 9(4):437–482, 1999.
3 Thomas Ehrhard. Finiteness spaces. *Mathematical Structures in Computer Science*, 15(4):615–646, 2005.
4 Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1-3):1–41, 2003.
5 Thomas Ehrhard and Laurent Regnier. Differential interaction nets. *Theoretical Computer Science*, 364(2):166–195, 2006.
6 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

* This work was partially supported by the ANR project CHOCO ANR-07-BLAN-0324.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 1–1

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

The Freedoms of Guarded Bisimulation

Martin Otto

Department of Mathematics
Technische Universität Darmstadt
otto@mathematik.tu-darmstadt.de

Abstract

Guarded logics have been shown to be amazingly versatile and tractable logics since their inception by Andréka, van Benthem, Németi in [1]. Features otherwise known from modal logics are lifted to the richer setting of general relational structures. The leading idea in this generalisation is the relativisation of quantifiers to tuples that are guarded by relational atoms. In a sense that can be made precise for many specific issues, guarded logics relate to general relational structures and hypergraphs in much the same way that modal logics relate to Kripke structures and graphs. This is particularly apparent for the associated semantic equivalence games. Guarded bisimulation can be seen as derived from a hypergraph version of ordinary (modal) bisimulation for graph-like structures. Just like preservation under ordinary bisimulation accounts for much of the good model-theoretic behaviour of modal logics, so hypergraph bisimulation and guarded bisimulation are the keys to understanding the model theory of guarded logics [1, 3, 4]. Model constructions and transformations that are compatible with guarded bisimulation account for the malleability of models and the tractability of the finite and algorithmic model theory of various guarded logics. They can often be cast in terms of hypergraph constructions [5, 6]. Here I intend to survey and summarise a number of results in the light of such model constructions including some more recent developments from [2, 6]. Results to be surveyed include finite and small model properties, decidability results, complexity and expressive completeness issues. At the conceptual as well as at the technical level, guarded bisimulations and local versus global properties in the overlap patterns of (finite) hypergraph structures form one of the leading themes, and offer intriguing combinatorial challenges.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Model theory, guarded logic, bisimulation, hypergraphs

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.2

References

1 H. Andréka, J. van Benthem and I. Németi, *Modal languages and bounded fragments of predicate logic*, Journal of Philosophical Logic, 27 (1998), pp. 217–274.
2 V. Barany and G. Gottlob and M. Otto, *Querying the guarded fragment*, Proc. 25th IEEE Symposium on Logic in Computer Science, 2010, pp. 1–11. Preprint of journal version (submitted to LMCS) available online.
3 E. Grädel, *On the restraining power of guards*, Journal of Symbolic Logic, 64 (1999), pp. 1719–1742.
4 E. Grädel, C. Hirsch and M. Otto, *Back and forth between guarded and modal logics*, ACM Transactions on Computational Logic, 3 (2002), pp. 418–463.
5 I. Hodkinson and M. Otto, *Finite conformal hypergraph covers and Gaifman cliques in finite structures*, The Bulletin of Symbolic Logic, 9 (2003), pp. 387–405.
6 M. Otto, *Highly acyclic groups, hypergraph covers and the guarded fragment*, Proc. 25th IEEE Symposium on Logic in Computer Science, 2010, pp. 12–21. Preprint of journal version (to appear in JACM) available online.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 2–2

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

Branching vs. Linear Time: Semantical Perspective*

Moshe Y. Vardi

Department of Computer Science, Rice University
Houston, TX 77005, USA
vardi@cs.rice.edu.com

Abstract

The discussion of the relative merits of linear- versus branching-time frameworks goes back to the early 1980s. One of the beliefs dominating this discussion has been that the linear-time framework is not expressive enough semantically, making linear-time logics lacking in expressiveness. In this talk we examine the branching-linear issue from the perpsective of process equivalence, which is one of the most fundamental notions in concurrency theory. Defining a notion of process equivalence essentially amounts to defining semantics for processes. Over the last three decades numerous notions of process equivalence have been proposed. Researchers in this area do not anymore try to identify the "right" notion of equivalence. Rather, focus has shifted to providing taxonomic frameworks, such as "the linear-branching spectrum", for the many proposed notions and trying to determine suitability for different applications.

We revisit here this issue from a fresh perspective. We postulate three principles that we view as fundamental to any discussion of process equivalence. First, we borrow from research in denotational semantics and take contextual equivalence as the primary notion of equivalence. This eliminates many testing scenarios as either too strong or too weak. Second, we require the description of a process to specify all relevant behavioral aspects of the process. Finally, we require the observable behavior of a process to be reflected in its input/output behavior. Under these principles, linear-time semantics emerges as the right approach. As an example, we apply these principles to the framework of transducers, a classical notion of state-based process that dates back to the 1950s and is well suited to reactive systems. We show that our principles yield a unique notion of process equivalence, which is trace based, rather than tree based.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Linear time, branching time, process equivalence, contextual equivalence

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.3

References

1 S. Nain and M.Y. Vardi. Branching vs. linear time: Semantical perspective. In *5th Int. Symp. on Automated Technology for Verification and Analysis*, volume 4762 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2007.

2 S. Nain and M.Y. Vardi. Trace semantics is fully abstract. In *Proc. 24th IEEE Symp. on Logic in Computer Science*, pages 59–68. IEEE Computer Society, 2009.

3 M.Y. Vardi. Branching vs. linear time: Final showdown. In *Proc. 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2001.

* Work supported in part by NSF grants CCF-0728882, and CNS 1049862, by BSF grant 9800096, and by a gift from Intel.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 3–3

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

Ontology-Based Data Access and Constraint Satisfaction

Frank Wolter

University of Liverpool, Department of Computer Science
Ashton Building, United Kingdom
wolter@liverpool.ac.uk

Abstract

In recent years, the use of ontologies (=logical theories) to access instance data has become increasingly popular. The general idea is that an ontology provides a vocabulary or conceptual model for the application domain, which can then be used as an interface for querying instance data and to derive additional facts [2, 6].

In this presentation, I will introduce ontology-based data access for ontologies given in description logics and investigate the following non-uniform complexity problem: what is the data complexity of conjunctive query answering for a fixed ontology? I will present general conditions under which this problem is in PTime and, respectively, coNP-hard. Then it is shown that for the basic description logic $\mathcal{ALC}$ (=modal logic), conjunctive query answering is equivalent to solving constraint satisfaction problems with finite templates. Examples of consequences of this result include: (i) a P/coNP dichotomy holds for conjunctive query answering with $\mathcal{ALC}$ if, and only if, Feder and Vardi's dichotomy conjecture [3] for constraints satisfaction problems holds; (ii) first-order constraint satisfaction problems investigated and characterized in [4] correspond exactly to reductions of ontology based data access with $\mathcal{ALC}$ to standard query answering over relational databases known as FO-rewriting.

By employing results from [1], we also show that if functional relations are added to $\mathcal{ALC}$, then the word problem of every non-deterministic polynomial time Turing Machine can be reduced to conjunctive query answering. Thus, by Ladner's Theorem, ontologies with coNP-intermediate conjunctive query answering exist.

The talk is based on joint work with Carsten Lutz. Preliminary results are published in [5].

1998 ACM Subject Classification F.4.1 Computational Logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.4

References

1 F. Baader, M. Bienvenu, C. Lutz, and F. Wolter. *Query and predicate emptiness in description logics*. Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning (KR), 2010.

2 D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. *Tractable reasoning and efficient query answering in description logics: The DL-Lite family*. Journal of Automated Reasoning, 39(3):385–429, 2007.

3 T. Feder and M. Y. Vardi. *Monotone monadic SNP and Constraint Satisfaction*. STOC, pages 612–622, 1993

4 B. Larose, C. Loten, and C. Tardif. *A characterisation of first-order constraint satisfaction problems*. Logical Methods in Computer Science, 3(4), 2007.

5 C. Lutz and F. Wolter. *Non-Uniform Data Complexity of Query Answering in Description Logics*. Proceedings of Description Logic Workshop, CEUR Workshop Proceedings, 2011

6 C. Lutz, D. Toman, F Wolter. *Conjunctive Query Answering in the Description Logic EL Using a Relational Database System*. Proceedings of IJCAI, pages 2070-2075, 2009.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 4–4

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

Power-Set Functors and Saturated Trees*

Jiří Adámek[1], Stefan Milius[1], Lawrence S. Moss[2], and Lurdes Sousa[3]

1 **Institut für Theoretische Informatik, Technische Universität Braunschweig Germany**
2 **Department of Mathematics, Indiana University Bloomington, IN, USA**
3 **Departamento de Matemática, Instituto Politécnico de Viseu, Portugal**

Abstract

We combine ideas coming from several fields, including modal logic, coalgebra, and set theory. Modally saturated trees were introduced by K. Fine in 1975. We give a new purely combinatorial formulation of modally saturated trees, and we prove that they form the limit of the final ω^{op}-chain of the finite power-set functor $\mathscr{P}_f$. From that, we derive an alternative proof of J. Worrell's description of the final coalgebra as the coalgebra of all strongly extensional, finitely branching trees. In the other direction, we represent the final coalgebra for $\mathscr{P}_f$ in terms of certain maximal consistent sets in the modal logic K. We also generalize Worrell's result to M-labeled trees for a commutative monoid M, yielding a final coalgebra for the corresponding functor $\mathscr{M}_f$ studied by P. Gumm and T. Schröder. We introduce the concept of an i-saturated tree for all ordinals i, and then prove that the i-th step in the final chain of the power set functor consists of all i-saturated trees. This leads to a new description of the final coalgebra for the restricted power-set functors $\mathscr{P}_\lambda$ (of subsets of cardinality smaller than λ).

1998 ACM Subject Classification F.4.1 Mathematical Logic; F.3.2 Semantics of Programming Languages; G.2.2 Graph Theory

Keywords and phrases Saturated tree, extensional tree, final coalgebra, power-set functor, modal logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.5

1 Introduction

Final coalgebras play a fundamental rôle in the theory of systems represented as coalgebras: J. Rutten [18] demonstrated that the final coalgebra describes all possible behaviors of states of systems. For Kripke structures considered as the coalgebras for the finite power-set functor $\mathscr{P}_f$ two beautiful descriptions of the final coalgebra exist: as the set of all hereditarily finite sets in the non-wellfounded set theory due to P. Aczel, see [2], and as the set of all strongly extensional, finitely branching trees[1] due to J. Worrell [21]. He used metric spaces: he described the limit $\mathscr{P}_f^\omega 1$ of the final chain of $\mathscr{P}_f$ as the set of all strongly extensional, compactly branching trees. From that he derived the above description of the final coalgebra. We give below two new descriptions that do not need topology, one combinatorial and one

* Extended Abstract

[1] Throughout the paper trees are directed graphs with a distinguished node called the root from which every other node can be reached by a unique directed path, and they are always considered up to isomorphism. Strong extensionality for trees is recalled in Section 2 below.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 5–19

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

using modal logic. We prove that the limit $\mathscr{P}_f^\omega 1$ consists (a) of all saturated trees or (b) of all maximal consistent theories of the modal logic K. And an alternative description of the final coalgebra is: the set of all hereditarily finite (maximal consistent) theories. Related descriptions were provided by S. Abramsky [1], A. Kurz and D. Pattinson [14] and by J. Rutten [17, Theorem 7.4].

We also present a generalization in two directions: one uses finite multisets with multiplicities drawn from a given commutative monoid M, as introduced by H. P. Gumm and T. Schröder [12]. Form the functor $\mathscr{M}_f$ of all such finite multisets; its coalgebras are labeled transition systems with actions from $M \setminus \{0\}$. We prove a direct generalization for all monoids for which $\mathscr{M}_f$ preserves weak pullbacks: the final coalgebra for $\mathscr{M}_f$ consists of all finitely branching, strongly extensional M-labeled trees. For general monoids this result is not true, but we prove that the final coalgebra for $\mathscr{M}_f$ is the coalgebra of extensional, finitely branching M-labeled trees modulo an equivalence generalizing M. Barr's equivalence for $\mathscr{P}_f$, see [7].

The other direction of generalization of the final coalgebra for $\mathscr{P}_f$ is from finite subsets to subsets of cardinality less than λ, where λ is an infinite cardinal. The corresponding power-set functor $\mathscr{P}_\lambda$ has the final coalgebra of all strongly extensional λ-branching trees, as proved by D. Schwencke [19]. We present a different proof based on the description of the final chain $\mathscr{P}^i 1$ of the (full) power-set functor $\mathscr{P}$. We introduce the concept of an i-saturated tree for every ordinal i (where ω-saturated is the above concept), and we describe $\mathscr{P}^i 1$ as the set of all strongly extensional i-saturated trees.

2 Extensional and saturated trees

For an endofunctor H of **Set** recall that a *coalgebra* is a set A together with a morphism $a\colon A \to HA$. A *coalgebra homomorphism* into $b\colon B \to HB$ is a morphism $f\colon A \to B$ with $b{\cdot}f = Hf{\cdot}a$. The final coalgebra, if it exists, is denoted by νH; by Lambek's Lemma [15] its coalgebra structure is an isomorphism $\nu H \xrightarrow{\sim} H(\nu H)$. For example Kripke structures (W, R, l) where $R \subseteq W \times W$ and $l\colon W \to 2^{\mathsf{AP}}$ are precisely the coalgebras for $HX = \mathscr{P}X \times 2^{\mathsf{AP}}$ where AP is a fixed set of atomic propositions and $\mathscr{P}$ is the power-set functor. In the present paper we restrict ourselves to the case $\mathsf{AP} = \emptyset$. Then Kripke structures are simply graphs, or coalgebras for $\mathscr{P}$. And the finitely branching graphs are coalgebras for the finite power-set functor $\mathscr{P}_f$.

In this and the next section we describe the final coalgebra for $\mathscr{P}_f$. Lambek's Lemma implies that $\mathscr{P}$ does not have a final coalgebra, but we describe the final chain of $\mathscr{P}$ in Section 5.

Recall from [7], dualizing the initial chain of [4], the *final chain* of H which is the chain $W\colon \mathbf{Ord}^{\mathrm{op}} \to \mathbf{Set}$ determined (uniquely up-to natural isomorphism) by its objects W_i, $i \in \mathbf{Ord}$, and connecting morphisms $w_{i,j}\colon W_i \to W_j$ $(i \geq j)$ as follows $W_0 = 1$, $W_{i+1} = HW_i$, and $W_i = \lim_{j<i} W_j$ for limit ordinals i and $w_{i+1,j+1} = Hw_{i,j}$, whereas $(w_{i,j})_{j<i}$ is a limit cone for limit ordinals i. If this chain *converges* at some ordinal i, i.e., the connecting map $HW_i \to W_i$ is an isomorphism, then its inverse yields the final coalgebra for H. The finite steps of the final chain of H are called the *final ω^{op}-chain of H*.

▶ **Remark 2.1.** Recall that coalgebras for $\mathscr{P}$ are simply graphs. When speaking about morphisms between graphs (in particular trees) we always mean coalgebra homomorphisms $f : A \to B$. That is, f preserves edges, and for every edge from $f(a)$ to b in B there exists an edge from a to a' in A with $b = f(a')$. Quotients of graphs are, as usual, represented by epimorphisms, that is, surjective morphisms.

▶ **Definition 2.2.** A tree is *extensional* if distinct children of any node define non-isomorphic subtrees.

The *extensional modification* of a tree t is the smallest quotient of t which is extensional. It is obtained from t by recursively identifying isomorphic subtrees whose roots have a joint parent.

▶ **Example 2.3.** The extensional modification of the complete binary tree is the single path.

▶ **Notation 2.4.** For every tree t denote by $\partial_n t$ the extensional tree obtained by cutting t at level n (i.e. deleting all nodes of depth $> n$) and forming the extensional modification. For all trees t and u, we write $t \sim_n u$ to mean that $\partial_n t = \partial_n u$ (remember that we identify isomorphic trees).

▶ **Remark 2.5.** The final ω^{op}-chain of $\mathscr{P}_f$ can be described as follows:

$$\mathscr{P}_f^n 1 = \text{all extensional trees of depth} \leq n \text{ with the connecting maps } \partial_n \colon \mathscr{P}_f^{n+1} 1 \to \mathscr{P}_f^n 1.$$

Indeed, the unique element of 1 can be taken to be the root-only tree. Given a set $M \subseteq \mathscr{P}_f^n 1$, we identify it with the tree tupling of its elements and obtain a tree in $\mathscr{P}_f^{n+1} 1$. The first connecting map from $\mathscr{P}_f 1$ to 1 is obviously ∂_0, and given that the n-th connecting map is $\partial_n \colon \mathscr{P}_f^{n+1} 1 \to \mathscr{P}_f^n 1$, it follows that the next connecting map, $\mathscr{P}_f \partial_n$, is (with the above tree tupling identification) precisely ∂_{n+1}.

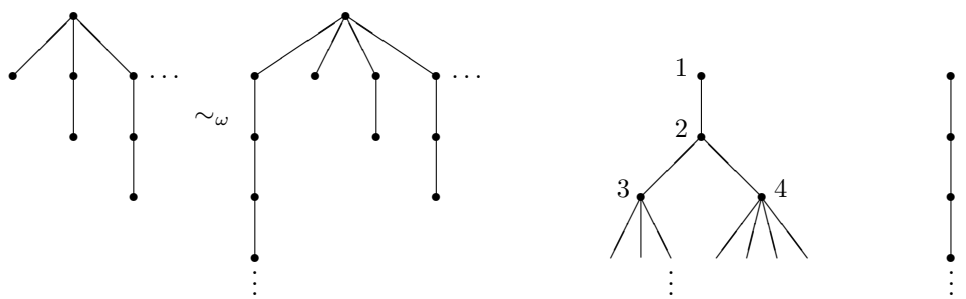
▶ **Definition 2.6.** Two trees t and u are called *Barr equivalent*, notation $t \sim_\omega u$, provided that $t \sim_n u$ holds for all $n < \omega$.

▶ **Remark 2.7.** The set B of all finitely branching extensional trees is a coalgebra for $\mathscr{P}_f$: the coalgebra map is the inverse of tree tupling. This coalgebra is weakly final, and a final coalgebra can be described as its quotient:

▶ **Theorem 2.8** (see [7]). *The final coalgebra for $\mathscr{P}_f$ can be described as the quotient $B/{\sim_\omega}$ of the coalgebra of all finitely branching, extensional trees modulo Barr-equivalence.*

▶ **Definition 2.9** (see [21])**.** For trees t and s a *tree bisimulation* is a relation $R \subseteq t \times s$ such that the roots are related, two related child nodes always have related parents, and R is a bisimulation w.r.t. $\mathscr{P}$; i.e., given related nodes $a\ R\ b$ then for every child a' of a in t there exists a child b' in s with $a'\ R\ b'$, and vice versa.

▶ **Example 2.10.** The first two trees in the picture below are Barr-equivalent trees. The third and fourth trees are two extensional trees (the third tree has n children of the n-th node) that are bisimilar. In fact, every tree without leaves is bisimilar to the infinite path.



▶ **Definition 2.11** (see [21]). A tree t is called *strongly extensional* if distinct children of any node are not bisimilar. Equivalently, every tree bisimulation $R \subseteq t \times t$ satisfies $R \subseteq \Delta_t$, where Δ_t is the diagonal relation on t.

▶ **Example 2.12.** (1) Every finite extensional tree is strongly extensional.

(2) The infinite path is a strongly extensional tree. This is the only strongly extensional tree without leaves: for every tree t without leaves the relation

$$x \; R \; y \quad \text{iff } x \text{ and } y \text{ have the same depth}$$

is a tree bisimulation. Thus, the third tree in Example 2.10 shows an extensional tree which is not strongly extensional.

▶ **Definition 2.13.** Given a tree t, the subtree of t rooted at the node x is denoted by t_x. A tree t is called *saturated* provided that for all nodes x of t and all trees s, if for all n, there are children x_n of x with $s \sim_n t_{x_n}$ $(n < \omega)$, then there is some fixed child y of x with $s \sim_\omega t_y$.

▶ **Example 2.14.** (1) Every finite tree is saturated.

(2) More generally: all finitely branching trees are saturated. Indeed, given x_n as above, there exists $k < \omega$ with $x_n = x_k$ for infinitely many n, and then $s \sim_n t_{x_k}$ for infinitely many n, proving $s \sim_\omega t_{x_k}$.

(3) The left-hand tree of Example 2.10 is not saturated. We obtain a saturated tree by adding a new child whose subtree is the infinite path.

(4) For every set $A \subseteq \omega$ the following tree r_A is saturated and strongly extensional: take an infinite path and add a leaf at depth n iff $n+1$ lies in A. We know from item (2) that r_A is saturated, and strong extensionality is obvious.

▶ **Lemma 2.15.** *Given saturated, strongly extensional trees t and u with $t \sim_\omega u$, we have $t = u$. Therefore there exist precisely $2^{\aleph_0}$ saturated, strongly extensional trees and they have branching at most $2^{\aleph_0}$.*

Proof. (a) For every child z of the root of u there exists a child y of the root of t with $t_y \sim_\omega u_z$. Indeed, since $t \sim_n u$ for every n there exist children x_n with $t_{x_n} \sim_n u_z$ $(n < \omega)$ and then y exists since t is saturated. Conversely, for every y there exists z with $t_y \sim_\omega u_z$. By continuing to lower nodes we conclude that the relation $R \subseteq t \times u$ defined recursively by

$$y \; R \; z \qquad \text{iff} \qquad \begin{cases} \text{either } y \text{ and } z \text{ are the roots} \\ \text{or } y \text{ and } z \text{ have } R\text{-related parents and } t_y \sim_\omega u_z \end{cases}$$

is a tree bisimulation. Clearly, the opposite relation R^{op} is a tree bisimulation, too, and, as $\mathscr{P}_f$ preserves weak pullbacks, so are the composite relations $R^{\mathrm{op}} \circ R \subseteq t \times t$ and $R \circ R^{\mathrm{op}} \subseteq u \times u$. Since t and u are strongly extensional, we conclude $R^{\mathrm{op}} \circ R \subseteq \Delta_t$ and $R \circ R^{\mathrm{op}} \subseteq \Delta_u$. Finally, since R and R^{op} are total relations, the last two inequalities are equalities, and this implies that R is the graph of an isomorphism from t to u, i.e., $t = u$.

(b) The number of saturated, strongly extensional trees is at least $2^{\aleph_0}$ by Example 2.14(4). It is at most $2^{\aleph_0}$ because every saturated strongly extensional tree t is determined by the set $M = \{t_x; x \text{ a child of the root of } t\}$, and M is determined, due to (a), by the sequence of sets $M_n = \{\partial_n s; s \in M\}$ for $n < \omega$. Since M_n is finite, the number of these sequences is at most $2^{\aleph_0}$.

The last statement follows since every subtree of a saturated tree is saturated. ◀

Recall Worrell's description of the limit $\mathscr{P}_f^\omega 1 = \lim_{n<\omega} \mathscr{P}_f^n 1$ of the final chain of $\mathscr{P}_f$ as the set of all compactly branching trees [21]. Here is a new combinatorial description:

▶ **Theorem 2.16.** *The limit $\mathscr{P}_f^\omega 1$ of the final ω^{op}-chain of $\mathscr{P}_f$ can be described as the set of all saturated, strongly extensional trees. The limit cone is $(\partial_n)_{n<\omega}$.*

Proof. Let S be the set of all saturated, strongly extensional trees. We prove that $\partial_n \colon S \to \mathscr{P}_f^n 1$ (see Remark 2.5) is a limit cone. For definiteness, we denote the connecting morphism of the final ω^{op}-chain by $\partial'_n \colon \mathscr{P}_f^{n+1} 1 \to \mathscr{P}_f^n 1$. It is obvious that $\partial_n = \partial'_n \cdot \partial_{n+1}$, thus (∂_n) is a cone on the final ω^{op}-chain.

(a) The cone ∂_n is collectively monic by Lemma 2.15.

(b) For every compatible family $r^n \in \mathscr{P}_f^n 1$ we prove that there exists $t \in S$ with $\partial_n t = r^n$ for every n. Compatibility means $r^n = \partial'_n(r^{n+1})$ for $n < \omega$. Let $\hat{r}^{n+1}$ be the tree obtained by cutting r^{n+1} at depth n. Since r^n is extensional, the above equation tells us that r^n is a quotient of $\hat{r}^{n+1}$. Let $e_n \colon \hat{r}^{n+1} \to r^n$ be the corresponding epimorphism. Define a tree t to have as nodes of depths $k = 0, 1, 2 \dots$ all sequences $\bar{x} = (\bar{x}^k, \bar{x}^{k+1}, \bar{x}^{k+2} \dots)$ of nodes $\bar{x}^n \in r^n$ of depth k with $e_n(\bar{x}^{n+1}) = \bar{x}^n$ for all $n \geq k$. Thus the sequence of roots of $r^0, r^1, r^2 \dots$ is the root of t. And edges are defined componentwise: there is an edge from $(\bar{x}^k, \bar{x}^{k+1}, \bar{x}^{k+2} \dots)$ to $(\bar{y}^{k+1}, \bar{y}^{k+2}, \bar{y}^{k+3}, \dots)$ iff $(\bar{x}^n, \bar{y}^n)$ is an edge of r^n for all $n \geq k+1$. It is easy to verify that t is a well-defined tree.

(b1) We prove $\partial_n t = r^n$. To this end it suffices to establish that there is an epimorphism of graphs from the cutting of t at level n to r^n (the desired equality then follows since r^n is extensional). Consider the n-th projection. This is surjective:

For every node $z \in r^n$ there exists a node $\bar{x} \in t$ with $z = \bar{x}^n$. Indeed, put $\bar{x}^n = z$, and since e_n is an epimorphism, choose $\bar{x}^{n+1} \in e_n^{-1}(\bar{x}^n)$, etc. Then $\bar{x} = (\bar{x}^n, \bar{x}^{n+1}, \bar{x}^{n+2} \dots)$ has the required property.

It is clear that the projection is a graph morphism: it preserves edges by the definition of t. And analogously to the argument of surjectivity above, for every $\bar{x}$ in t and every edge from $z = \bar{x}^n$ to z' in r^n there exists an edge from $\bar{x}$ to x' in t with $z' = (\bar{x}')^n$.

(b2) t is strongly extensional. Indeed, given a tree bisimulation $R \subseteq t \times t$, we prove that $\bar{x}\ R\ \bar{y}$ implies $\bar{x} = \bar{y}$. Let k be the depth of $\bar{x}$ and $\bar{y}$. From (b1) it follows that $r^n_{\bar{x}^n} = \partial_n(t_{\bar{x}})$ and $r^n_{\bar{y}^n} = \partial_n(t_{\bar{y}})$ for all $n \geq k$. But $\bar{x}\ R\ \bar{y}$ implies that R restricts to a tree bisimulation between $t_{\bar{x}}$ and $t_{\bar{y}}$, thus, $t_{\bar{x}} \sim_n t_{\bar{y}}$. Consequently, $r^n_{\bar{x}^n} = r^n_{\bar{y}^n}$. This implies $\bar{x}^n = \bar{y}^n$. (This is clear from extensionality of r^n in case $k = 1$. This finishes the proof of $\bar{x} = \bar{y}$ if $k = 1$. For $k = 2$, we conclude that $\bar{x}$ and $\bar{y}$ have the same parent, $\bar{z}$, and apply the above to $t_{\bar{z}}$ in lieu of t, etc.)

(b3) The tree t is saturated. Indeed, let s be a tree for which the condition of Definition 2.13 holds, taking x to be the root of t. (The proof for all other nodes x of t is completely analogous.) That is, we have children $\bar{x}_n$ of the root of t with $s \sim_n t_{\bar{x}_n}$ for $n < \omega$. We prove that the node $\bar{y}$ of t with components $\bar{y}^n = \big(\bar{x}_n\big)^n$ for all $n \geq 1$ fulfils $s \sim_\omega t_y$.

Firstly, we need to verify that $\bar{y}$ is a node: $e_n(\bar{y}^{n+1}) = \bar{y}^n$. Both of these nodes are children of the root of r^n, thus, by extensionality we only need to prove that they define the same subtree of r^n. Let $\hat{t}_{\bar{x}_n}$ be the cutting of $t_{\bar{x}_n}$ at level $n-1$, then from (b1) we know that $r^n_{\bar{y}^n}$ is the image of $\hat{t}_{\bar{x}_n}$ under the n-th projection $t \to r^n$. Since $r^n_{\bar{y}_n}$ is extensional, this proves $\partial_n t_{\bar{x}_n} = r^n_{\bar{y}_n}$ and analogously for $n+1$. Moreover, from $t_{\bar{x}_n} \sim_n s \sim_{n+1} t_{\bar{x}_{n+1}}$ we deduce $t_{\bar{x}_n} \sim_n t_{\bar{x}_{n+1}}$. Consequently,

$$r^n_{\bar{y}^n} = \partial_n t_{\bar{x}_{n+1}} \qquad \text{and} \qquad r^{n+1}_{\bar{y}^{n+1}} = \partial_{n+1} t_{\bar{x}_{n+1}}.$$

We also have $\partial'_n r^{n+1}_{\bar{y}^{n+1}} = r^n_{e_n(\bar{y}^{n+1})}$ because the right-hand tree is extensional, and it is the

image of $\hat{r}^{n+1}_{\bar{y}^{n+1}}$ under e_n. Consequently, $r^n_{e_n(\bar{y}^{n+1})} = \partial'_n \partial_{n+1} t_{\bar{x}_{n+1}} = \partial_n t_{\bar{x}_{n+1}}$. This proves $r^n_{e_n(\bar{y}^{n+1})} = r^n_{\bar{y}_n}$, thus, $e_n(\bar{y}^{n+1}) = \bar{y}^n$ by extensionality of r^n.

Next, we need to verify $s \sim_n t_{\bar{y}}$ for every $n < \omega$. Indeed, we have $s \sim_n t_{\bar{x}_n}$, and to prove $t_{\bar{x}_n} \sim_n t_{\bar{y}}$ observe that the n-th projection $t \to r^n$ maps the cutting of $t_{\bar{y}}$ onto $r^n_{\bar{y}^n}$, thus, $r^n_{\bar{y}^n} = \partial t_{\bar{y}}$. We already observed that $r^n_{\bar{y}^n} = \partial_n t_{\bar{x}_n}$, thus, $\partial_n t_{\bar{x}_n} = \partial_n t_{\bar{y}}$. ◀

▶ **Corollary 2.17** (J. Worrell). *The final chain of $\mathscr{P}_f$ converges in $\omega + \omega$ steps with the step $\omega + n$ given by the set $\mathscr{P}_f^{\omega+n}1$ of all saturated, strongly extensional trees finitely branching up to level $n-1$. Moreover, the final coalgebra for $\mathscr{P}_f$ is given by*

$$\mathscr{P}_f^{\omega+\omega}1 = \textit{all finitely branching, strongly extensional trees.}$$

Indeed, for $n = 1$ we have $\mathscr{P}_f^{\omega+1} = \mathscr{P}_f(\mathscr{P}_f^{\omega}1)$ and we identify, again, every finite set $M \subseteq \mathscr{P}_f^{\omega}1$ of saturated trees with its tree-tupling. This is, by Example 2.14(2), a saturated, strongly extensional tree which is finitely branching at the root—and conversely, every such tree is a tree tupling of a finite subset of $\mathscr{P}_f^{\omega}1$. Analogously for $n = 2$: we have $\mathscr{P}_f^{\omega+2} = \mathscr{P}_f(\mathscr{P}_f^{\omega+1}1)$ and the resulting trees are precisely those trees in $\mathscr{P}_f^{\omega}1$ that are finitely branching at levels 0 and 1, etc. The connecting maps are the inclusion maps. The limit $\mathscr{P}_f^{\omega+\omega}1 = \lim_{n<\omega} \mathscr{P}_f^{\omega+n}1$ is the intersection of these subsets of $\mathscr{P}_f^{\omega}1$ which consists of all finitely branching, strongly extensional trees: they are saturated, see Example 2.14(2).

3 Modally saturated trees

K. Fine [10] introduced the concept of modal saturatedness for Kripke structures in modal logic. In this section, we review all of the needed definitions, and we prove that modally saturated trees are the same as saturated trees.

(a) We work with modal logic formulated *without* atomic propositions. The sentences φ of modal logic are then given by

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box\varphi$$

We use the usual abbreviations:

$$\bot = \neg\top \qquad \varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi) \qquad \varphi \to \psi = \neg\varphi \vee \psi \qquad \Diamond\varphi = \neg\Box\neg\varphi.$$

A sentence has *depth* n if n is the maximum of nested $\Box$ in it.

(b) We interpret modal logic on Kripke structures. Since we have no atomic sentences, our Kripke structures are just graphs $G = (G, \to)$, where $\to$ is a binary relation on the set G. The main semantic relation is the *satisfaction* relation $\models$ between the node set of a given graph and the sentences of the logic. This is defined as follows:

$a \models \top$		always
$a \models \neg\varphi$	iff	it is not the case that $a \models \varphi$
$a \models \varphi \wedge \psi$	iff	$a \models \varphi$ and $a \models \psi$
$a \models \Box\varphi$	iff	for all neighbors b of a, $b \models \varphi$

Given a tree t we write $t \vDash \varphi$ if the root satisfies φ.

(c) A *theory* is a set S of sentences. We write $a \vDash S$ if $a \vDash \varphi$ for all $\varphi \in S$ and call a a *model* of S.

(d) Turning to the proof system, the modal logic K extends the propositional logic (Hilbert's style) by one axiom $\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$, called K, and one deduction rule: if $\varphi \in \mathsf{K}$ then $\Box\varphi \in \mathsf{K}$. We write $\vdash \varphi$ if φ can be derived in this logic.

This logic is sound and complete. That is, $\vdash \varphi$ holds iff for every node a of any graph, $a \vDash \varphi$.

(e) A theory S is *inconsistent* if for some finite $\{\varphi_1, \ldots, \varphi_n\} \subseteq S$, $\vdash \neg \bigwedge \varphi_i$. S is *consistent* if S is not inconsistent. Or, equivalently, S has a model. If, moreover, $S \cup \{\varphi\}$ is inconsistent for every sentence $\varphi \notin S$, then S is *maximal consistent*.

(f) $\Box S$ denotes the theory $\{\Box\varphi : \varphi \in S\}$, and $\Box^k S = \Box(\Box^{k-1} S)$ for $k \geq 2$.

▶ **Definition 3.1.** We define *canonical sentences* χ *of depth* n by recursion on n, as follows:
(a) $\top$ is the only canonical sentence of depth 0, and
(b) canonical sentences of depth $n+1$ are precisely the sentences

$$\nabla S = (\bigwedge \Diamond S) \wedge \Box \bigvee S$$

where S is a set of canonical sentences of depth n. We use the conventions that $\bigwedge \emptyset = \top$, $\bigvee \emptyset = \bot$, and we often identify sentences φ and ψ when $\vdash \varphi \leftrightarrow \psi$ in K.

▶ **Example 3.2.** We have two canonical sentences of depth 1.

$$\nabla\emptyset = \top \wedge \Box \bot = \Box\bot \qquad \text{and} \qquad \nabla\{\top\} = \Diamond\top \wedge \Box\top = \Diamond\top$$

distinguishing whether the given node has a neighbor or not.

▶ **Theorem 3.3** (K. Fine [10] and L. Moss [16])**.** *For every node a of a graph and every $n \in \mathbb{N}$ there exists a unique canonical sentence χ of depth n satisfied by a. Moreover, for every canonical sentence χ of depth n and every sentence ψ of depth at most n, either $\vdash \chi \to \psi$ or $\vdash \chi \to \neg\psi$.*

▶ **Notation 3.4.** (a) For every tree t we denote by $\chi_n(t)$ the unique canonical sentence of depth n satisfied in the root. It is easy to prove that

$$\chi_{n+1}(t) = \nabla\big\{\chi_n(t_x) : x \text{ child of the root of } t\big\}.$$

(b) For any graph G, and any $a \in G$, we denote by S_a the set of all sentences φ with $a \vDash \varphi$ in G. For a tree t, we similarly denote by S_t the set of sentences satisfied by the root of t.

(c) Recall from [8] that the *canonical model of* K is the graph C whose nodes are the maximal consistent theories, and with $S \to S'$ iff $\Diamond S' \subseteq S$ (equivalently, $\Box S \subseteq S'$). The *Truth Lemma* (see [8], Lemma 4.21) is the statement that for all $S \in C$,

$$\{\varphi : S \models \varphi \text{ in } C\} = S.$$

This lemma is easy to check by induction on φ.

▶ **Corollary 3.5.** *For two trees t and s we have $t \sim_n s$ iff $t \vDash \chi_n(s)$. Consequently, $t \sim_\omega s$ iff $S_t = S_s$.*

▶ **Proposition 3.6.** *The limit $\mathscr{P}_f^\omega 1$ can be described as the set C of all maximal consistent theories in K.*

Proof. We have described $\mathscr{P}_f^\omega 1$ as the set of all saturated, strongly extensional trees. We prove that $t \mapsto S_t$ is a bijection between this set and C. This finishes the proof. (a) For every $t \in \mathscr{P}_f^\omega 1$ the theory S_t is maximal consistent. Indeed, it is is obviously consistent. Given $\varphi \notin S$ of depth n, we have $t \nvDash \varphi$ and $t \vDash \chi_n(t)$, thus, $\nvdash \chi_n(t) \to \varphi$. By Theorem 3.3 $\vdash \chi_n(t) \to \neg\varphi$. Therefore, $S_t \cup \{\varphi\}$ is inconsistent. (b) By the Truth Lemma, every maximal consistent theory S is of the form S_t for some t: let t be the expansion of the canonical graph C at S. Moreover, t can be taken as saturated and strongly extensional, since the saturation operation on trees preserves modal theories (see Corollary 3.5). ◀

▶ **Definition 3.7.** A theory S is called *hereditarily finite* if it is maximal consistent and for every $k \in \mathbb{N}$ there exist only finitely many maximal consistent theories S' with $\Diamond^k S' \subseteq S$.

▶ **Theorem 3.8.** *The set of all hereditarily finite theories is a final coalgebra for $\mathscr{P}_f$ via the coalgebra map $S \mapsto \{S' : \Diamond S' \subseteq S\}$.*

Proof. We prove that the bijection $t \mapsto S_t$ of Proposition 3.6 has the property that for $t \in \mathscr{P}_f^\omega 1$ we have that t is finitely branching iff S_t hereditarily finite. From that our theorem follows, since the coalgebra map above corresponds to the coalgebra map of $\nu\mathscr{P}_f$. Indeed:

(a) If S_t is hereditarily finite, then t is finitely branching. It is sufficient to verify that t is finitely branching at the root. Given a node x of depth k, we then apply this to t_x: the theory of this subtree is also hereditarily finite, since $\Diamond^k S_{t_x} \subseteq S_t$ (indeed: if $t_x \vDash \varphi$ then $t \vDash \Diamond^k \varphi$).

Every child a of the root of t fulfils $\Diamond S_{t_a} \subseteq S_t$. Thus, there are only finitely many such theories S_{t_a}. Now let a and b be children of the root of t with $S_{t_a} = S_{t_b}$, whence $t_a \sim_\omega t_b$ by Corollary 3.5. So since t_a and t_b are saturated and strongly extensional, we have $t_a = t_b$ by Lemma 2.15. Therefore, the root has only finitely many children.

(b) If t is finitely branching, then S_t is hereditarily finite. Indeed, for every maximal consistent theory S' with $\Diamond^k S' \subseteq S_t$ let s be a tree with $S' = S_s$ (see Proposition 3.6). Then for every $n \in \mathbb{N}$ we have $t \vDash \Diamond^k \chi_n(s)$, i.e., some node of t of depth k satisfies $\chi_n(s)$. Since we have only finitely many such nodes, one of them, say a, satisfies $\chi_n(s)$ for all n. That is, $t_a \sim_n s$ for $n \in \mathbb{N}$, hence, $S_{t_a} = S'$, see Corollary 3.5. Since we have only finitely many nodes a of depth k, we see that S_t is hereditarily finite. ◀

▶ **Definition 3.9** (see [10])**.** A graph is called *modally saturated* if for every node a, given a theory S such that

$$a \vDash \Diamond \bigwedge S_0 \qquad \text{for every finite } S_0 \subseteq S \tag{3.1}$$

there exists a neighbor b of a satisfying S.

▶ **Theorem 3.10.** *A tree is saturated iff it is modally saturated.*

Proof. (a) Let t be modally saturated. Let a be a node in t, and let s be a tree with the property that there exist children x_n of a with $s \sim_n t_{x_n}$ $(n < \omega)$. We prove $s \sim_\omega t_b$ for some child b. The theory S_s fulfils (3.1): given $S_0 \subseteq S_s$ finite, let n be the maximum of the depths of all $\psi \in S_0$; then $\vdash \chi_n(s) \to \psi$ for all $\psi \in S_0$ (see Theorem 3.3). By Corollary 3.5, $s \sim_n t_{x_n}$ iff $x_n \models \chi_n(s)$, and this implies $x_n \vDash \psi$ for all $\psi \in S_0$. Thus, $a \vDash \Diamond \bigwedge S_0$. Let b be a neighbor of a satisfying S_s. Then $t_b \vDash \chi_n(s)$ for all n; i.e., $s \sim_\omega t_b$ by Corollary 3.5.

(b) Let t be saturated. Let a be a node of t and S be a theory satisfying (3.1). For every natural number n define S_n to be a set of representatives of all $\psi \in S$ of depth at most n modulo logical equivalence in K. As a corollary of Theorem 3.3 one readily proves that there is only a finite set of sentences of depth at most n (up to logical equivalence). So we have that S_n is finite. By (3.1) we see that for every n, there exists a child b_n of a such that

$$b_n \vDash \psi \qquad \text{for all } \psi \in S_n.$$

It is our task to prove that a has a child b satisfying S.

Let v be the graph whose nodes are all canonical sentences χ of depth any $n = 0, 1, 2, \ldots$ such that $a \vDash \Diamond\chi$ and $\vdash \chi \to \psi$ for all $\psi \in S_n$. We make v a graph using the *converse* of logical implication in K. So the neighbors of the node χ are all the nodes χ' of depth $n+1$

with $\vdash \chi' \to \chi$. The root is $\top$, and every node χ' of v has indeed a unique parent (so v is a tree): since $a \vDash \Diamond\chi'$, we have a child c of a with $c \vDash \chi'$ which by Theorem 3.3 implies $\chi' = \chi_{n+1}(t_c)$. Put $\chi = \chi_n(t_c)$, then $\vdash \chi' \to \chi$. (This is because $\vdash \chi' \to \neg\chi$ cannot happen due to $c \vDash \chi'$ and $c \vDash \chi$. Now use Theorem 3.3). Consequently, χ is a parent of χ'. And the uniqueness of the parent is obvious: suppose $\vdash \chi \to \chi'$ where $\chi' \in v$ has depth n, then $t_c \vDash \chi'$, therefore $\chi' = \chi_n(t_c)$.

The tree v is obviously finitely branching. And since each $\chi_n(t_{b_n})$ lies in v and each of these formulas has a different depth, they form an infinite set of nodes of v. By König's Lemma, v has an infinite branch

$$\top = \chi_0 \leftarrow \chi_1 \leftarrow \chi_2 \dots$$

Each $S \cup \{\chi_n\}$ is consistent. Indeed, by compactness it is sufficient to verify that $S_k \cup \{\chi_n\}$ is consistent for every $k \geq n$: due to $a \vDash \Diamond\chi_k$ we have a child c of a satisfying χ_k, then t_c is a model of S_k (due to $\vdash \chi_k \to \psi$ for all $\psi \in S_k$) and of χ_n (due to $\vdash \chi_k \to \chi_n$). Consequently, $S \cup \{\chi_0, \chi_1, \chi_2, \dots\}$ is consistent: use compactness again. Let s be a tree which is model of the last theory. Then $s \vDash \chi_n$ which by Theorem 3.3 implies $\chi_n = \chi_n(s)$ for every n. On the other hand, since $a \vDash \Diamond\chi_n$, we have a child c_n of a with $c_n \vDash \chi_n$, thus, $\chi_n = \chi_n(t_{c_n})$. By Corollary 3.5 this proves $s \sim_n t_{c_n}$. Since t is saturated, there exists a child b of a with $s \sim_\omega t_b$. Then $S \subseteq S_s = S_{t_b}$ which concludes the proof: b satisfies S. ◀

4 Finite multisets with multiplicities in a commutative monoid

Here we follow the approach of P. Gumm and T. Schröder [12] who investigated finitely branching Kripke structures with transitions having weights from a given commutative monoid $(M, +, 0)$. These are just coalgebras for the functor $\mathscr{M}_f\colon \mathbf{Set} \to \mathbf{Set}$ (denoted by $\mathscr{M}_\omega$ in [12]) assigning to every set X the set $\mathscr{M}_f X$ of all finite multisets in X, i.e. all functions $A : X \to M$ with $A^{-1}[M \setminus \{0\}]$ finite. Given a function $h\colon X \to Y$, the functor $\mathscr{M}_f$ assigns to every finite multiset $A\colon X \to M$ the finite multiset $\mathscr{M}_f h(A)$ sending $y \in Y$ to $\sum_{x \in X, h(x)=y} A(x)$.

▶ **Example 4.1.** The Boolean monoid $P = \{0, 1\}$ yields the finite power-set functor $\mathscr{P}_f$. The cyclic group $C = \{0, 1\}$ yields a functor $\mathcal{C}_f$ which coincides with $\mathscr{P}_f$ on objects but is very different on morphisms.

▶ **Definition 4.2.** By an *M-labeled graph* G is meant a graph whose edges are labeled in $M \setminus \{0\}$. We denote by $w_G\colon G \times G \to M$ the corresponding "weight" function with $w_G(x, y) \neq 0$ iff y is a neighbor of x.

▶ **Remark 4.3.** (a) The coalgebras for $\mathscr{M}_f$ are precisely the finitely branching M-labeled graphs. Indeed, given such a graph G, define the coalgebra structure $G \to \mathscr{M}_f G$ by assigning to every vertex x the finite multiset $w_G(x, -)\colon G \to M$. Conversely, every finitely branching M-labeled graph is obtained from precisely one coalgebra of $\mathscr{M}_f$.

(b) Coalgebra homomorphisms between two finitely branching M-labeled graphs G and H are precisely the functions $f\colon G \to H$ between the vertex sets such that

$$w_H\big(f(x), y\big) = \sum_{x' \in X, f(x')=y} w_G(x, x') \qquad \text{for all } x \in G,\ y \in H. \tag{4.1}$$

(c) We identify, once again, two M-labeled trees whenever they are isomorphic (as coalgebras for $\mathscr{M}_f$).

▶ **Definition 4.4.** An M-labeled tree is *extensional* if distinct children of any node define non-isomorphic M-labeled subtrees.

We use $\sim_n$ and $\sim_\omega$ in an obvious analogy to Notation 2.4 and Definition 2.6.

▶ **Remark 4.5.** The concepts of tree bisimulation and strong extensionality (see Definitions 2.9 and 2.11) also immediately generalize to M-labeled trees. We now generalize Theorem 2.8, using B again for the coalgebra of all finitely branching extensional M-labeled trees.

▶ **Theorem 4.6.** *Let M be a commutative monoid. The coalgebra $B/\!\sim_\omega$ of all finitely branching, extensional, M-labeled trees modulo Barr equivalence is final for $\mathscr{M}_f$.*

Sketch of proof. (1) B is weakly final. Indeed, for every finitely branching M-labeled graph (A, α) we define a coalgebra homomorphism $h\colon A \to B$ by assigning to every vertex $a \in A$ the extensional modification of the tree expansion of a. Recall that the nodes of the tree expansion of a are the paths $a_0, a_1, \ldots, a_k$ of A starting in a, including the empty path, a, which is the root. A child of $a_1, \ldots, a_k$ is any extension $a_1, \ldots, a_k, a_{k+1}$ and its weight in the tree expansion of a is $w_G(a_k, a_{k+1})$, see Definition 4.2.

(2) The final coalgebra is obtained from B by the quotient modulo the largest congruence. This follows from the fact that the category of coalgebras for $\mathscr{M}_f$ is complete. Thus, by Freyd's Adjoint Functor Theorem a weakly final object always has the final object as the quotient modulo the greatest congruence.

(3) The Barr equivalence is a congruence on B. That is, the quotient $B/\!\sim_\omega$ carries a coalgebra structure for $\mathscr{M}_f$ such that the quotient map $q\colon B \to B/\!\sim_\omega$ is a coalgebra homomorphism.

(4) Every congruence $\approx$ on B is contained in $\sim_\omega$. That is, our task is to prove the implication

$$t \approx t' \qquad \text{implies} \qquad \partial_n t = \partial_n t'.$$

This follows by induction on n since $\approx$ being a congruence means that for the quotient map $q\colon B \to B/\!\approx$ we have a coalgebra structure $B/\!\approx$ making q a homomorphism. ◀

▶ **Definition 4.7** (See [12])**.** A commutative monoid M is called

(a) *positive* if $a + b = 0$ implies $a = 0 = b$ and
(b) *refinable* if $a_1 + a_2 = b_1 + b_2$ implies that there exists a 2×2 matrix with row sums a_1 and a_2, respectively, and column sum b_1 and b_2, respectively.

▶ **Theorem 4.8.** *The following conditions on a commutative monoid M are equivalent*

(a) *The functor $\mathscr{M}_f$ weakly preserves pullbacks*
(b) *M is positive and refinable, and*
(c) *whenever $a_1 + \cdots + a_n = b_1 + \cdots + b_k$, there exists an $n \times k$-matrix whose vector of row sums is $a_1, \ldots, a_n$ and the vector of column sums is $b_1, \ldots, b_k$.*

In [12] this theorem is proved except that in lieu of (a) weak preservation of *non-empty* pullbacks is requested. However, the functor $\mathscr{M}_f$ has a unique distinguished point in the sense of V. Trnková [20], namely, the empty set $\emptyset \in \mathscr{M}_f X$. Since $\mathscr{M}_f \emptyset = \{\emptyset\}$, it follows from the result in [20] that $\mathscr{M}_f$ preserves weak pullbacks iff it preserves the nonempty ones. Now for (a) $\iff$ (b), see [12], Theorem 5.13, and concerning (b) $\iff$ (c), Proposition 5.10 of *loc. cit.* states that refinability is equivalent to condition (c) with $n, k > 1$ and positivity of M is equivalent to condition (c) with $n > 1$ and $k = 0$. For $n = 1$, condition (c) is trivial.

▶ **Example 4.9** (See [12]). The Boolean monoid $P = \{0,1\}$ and the monoids $(\mathbb{N},+,0)$ and $(\mathbb{N},\cdot,1)$ are positive and refinable. The cyclic group $\mathscr{C} = \{0,1\}$ is refinable but not positive. For every lattice L the monoid $\mathscr{L} = (L,\vee,0)$ is positive, and it is refinable iff L is a distributive lattice.

▶ **Theorem 4.10.** *Let M be a positive and refinable monoid. The coalgebra B_s of all strongly extensional, finitely branching M-labeled trees is final for $\mathscr{M}_f$.*

▶ **Remark 4.11.** For the coalgebra B of extensional M-labeled trees all strongly extensional trees clearly form a subcoalgebra $m\colon B_s \hookrightarrow B$. We prove that the composite of m with the quotient homomorphism $q\colon B \to B/{\sim_\omega}$ is an isomorphism $q{\cdot}m\colon B_s \to B/{\sim_\omega}$. This proves that B_s is final.

Sketch of proof. Since $q{\cdot}m$ is a homomorphism of coalgebras, it is sufficient to prove that it is a bijection, then it is an isomorphism. In other words: we are to prove that B_s is a choice class of $\sim_\omega$ on the set B.

(1) Every tree t in B is Barr equivalent to a strongly extensional tree $t/\bar{R}$. Indeed, recall from Theorem 4.8 that $\mathscr{M}_f$ weakly preserves pullbacks. Thus the greatest bisimulation $R \subseteq t \times t$ is an equivalence relation which is also the greatest congruence. And for the greatest tree bisimulation $\bar{R}$ contained in R, the strongly extensional tree $t/\bar{R}$ is bisimilar to t. Since $B/{\sim_\omega}$ is the final coalgebra, this proves that $t \sim_\omega t/\bar{R}$.

(2) If two strongly extensional trees are Barr equivalent, then they are equal. Instead, we prove in items (3) and (4) below that given trees $t, s \in B$ then if $t \sim_\omega s$ then t is bisimilar to s. Thus, we obtain a tree bisimulation $R \subseteq t \times s$ and, by symmetry, a tree bisimulation $S \subseteq s \times t$. Since $\mathscr{M}_f$ weakly preserves pullbacks, $S \circ R \subseteq t \times t$ is a tree bisimulation. This proves in the case t and s are strongly extensional, that $S \circ R = \Delta$. By symmetry, $R \circ S = \Delta$. Then R is a graph of an isomorphism from t to s, i.e., $t = s$.

(3) We consider the given trees $t \sim_\omega s$ as elements of the coalgebra B of Theorem 4.6. We know that $\sim_\omega$ is the greatest congruence, hence, the greatest bisimulation on B. As proved in [12], Lemma 5.5, this means that there exists a matrix $m\colon B \times B \to M$ such that

(a) $w_B(t,t') = \sum_{s' \in B} m(t',s')$ for all $t' \in B$

(b) $w_B(s,s') = \sum_{t' \in B} m(t',s')$ for all $s' \in B$, and

(c) $m(t',s') \neq 0$ implies $t' \sim_\omega s'$.

Since M is positive, whenever $m(t',s') \neq 0$, we have $w_B(t,t') \neq 0$, that is, there exists a child x of the root x_0 of t with $t' = t_x$ and $w_B(t,t') = w_t(x_0,x)$. Analogously, $m(t',s') \neq 0$ implies $s' = s_y$ for some child y of the root y_0 of s with $w_B(s,s') = w_s(y_0,y)$. Since t and s are extensional, the trees $t' \in B$ with $w_B(t,t') \neq 0$ are in bijective correspondence with the children x of x_0 in t via $x \mapsto t_x$. Analogously for s. Thus we can translate (a)–(c) as follows:

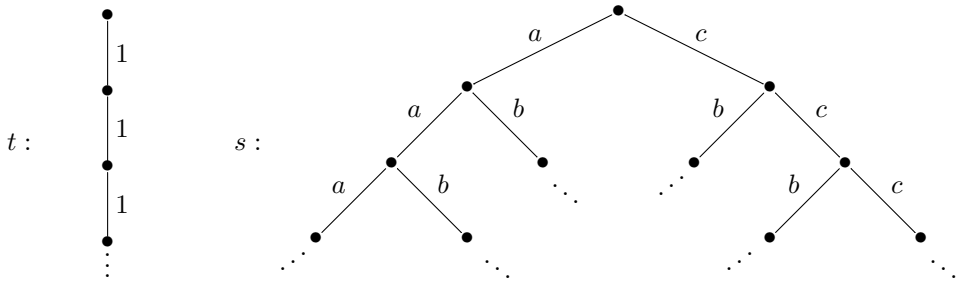
(a*) $w_t(x_0,x) = \sum_{y \in s} m(t_x,t_y)$ for all $x \in t$

(b*) $w_s(y_0,y) = \sum_{x \in t} m(t_x,t_y)$ for all $y \in s$, and

(c*) $m(t',s') \neq 0$ implies that there exists a unique child x of x_0 in t and a unique child y of y_0 in s with $t_x \sim_\omega t_y$, $t' = t_x$ and $s' = s_y$.

(4) We prove that given trees $\bar{t}, \bar{s} \in B$ with $\bar{t} \sim_\omega \bar{s}$, it follows that the relation $R \subseteq \bar{t} \times \bar{s}$ defined recursively by $x\ R\ y$ iff $\bar{t}_x \sim_\omega \bar{s}_y$ and x and y are roots or have R-related parents is a tree bisimulation. ◀

▶ **Example 4.12.** The above theorem does not generalize to all positive monoids. Indeed, consider the monoid $\mathscr{L} = (L, \vee, 0)$ for the lattice $\{0, a, b, c, 1\}$ where a, b, c are pairwise incomparable. Then strongly extensional finitely branching $\mathscr{L}$-labeled trees do not form a final coalgebra, since they are not a choice class of the Barr equivalence. The following trees are easily seen to be Barr equivalent:



Here s has as nodes the binary words, and the weights are, for all $x \in \{0,1\}^*$, defined by $w_s(x0, x00) = a$, $w_s(x1, x11) = c$ and $w_s(x0, x01) = b = w_s(x1, x10)$. It is obvious that t is strongly extensional. To prove that so is s, let $R \subseteq s \times s$ be a tree bisimulation. Using the conditions (a)–(c) in the preceding proof it is easy to verify that $R \subseteq \Delta_s$.

5 The final chain of $\mathscr{P}$

Although the power-set functor $\mathscr{P}$ has no final coalgebra, one can describe its final chain concretely: we now prove that the i-th step $\mathscr{P}^i 1$ consists of all i-saturated, strongly extensional trees. This generalization from saturated to i-saturated turns out the be quite nontrivial. In addition, this allows us to describe the final chain of the restricted power set functors $\mathscr{P}_\lambda$ (see Corollary 5.13).

▶ **Notation 5.1.** Recall that the subtree of t rooted at the node x is denoted by t_x. We define equivalences $\sim_i$ on the class of all trees for every ordinal i (cf. Notation 2.4 and Definition 2.6) by transfinite induction:

$s \sim_0 t$	holds for all pairs s, t;
$s \sim_{i+1} t$	holds iff for every child x of the root of s there is a child y of the root of t with $s_x \sim_i t_y$, and vice versa

and for limit ordinals i, $s \sim_i t$ means $s \sim_j t$ for all $j < i$.

▶ **Example 5.2.** $\sim_\omega$ is Barr equivalence, see Definition 2.6. The first and second trees in Example 2.10 are trees s and t with $s \nsim_{\omega+1} t$ which are Barr equivalent.

There exist, for every ordinal i, trees s and t with $s \sim_i t$ but $s \nsim_{i+1} t$, see [5].

▶ **Definition 5.3.** We define the concept of *i-saturated tree* for every ordinal i by transfinite induction: A tree t is i-saturated iff

(a) $i = 0$: t consists of the root only

(b) $i = j + 1$: t_x is j-saturated for every child x of the root

(c) i a limit ordinal: given a tree s and a node x of t having children x_j with $s \sim_j t_{x_j}$ $(j < i)$, then x has a child y with $s \sim_i t_y$.

▶ **Examples 5.4.** (a) For i finite, a tree is i-saturated iff it has height at most i. And $s \sim_i t$ holds iff $\partial_i s = \partial_i t$. Therefore, a tree is ω-saturated iff it is saturated in the sense of Definition 2.13.

(b) An example of an $(\omega+1)$-saturated tree which is not ω-saturated is the left-hand tree in Example 2.10.

(c) For every infinite cardinal λ, all λ-branching trees t are λ-saturated.

▶ **Remark 5.5.** For every tree s there exists a Barr-equivalent tree which is ω-saturated and strongly extensional. We denote it by $\partial_\omega s$ and call it the ω-saturation of s. In fact, for the sequence $r^n = \partial_n s$ of trees in $\mathscr{P}_f^n 1$, which is clearly compatible, apply the construction in the proof of Theorem 2.16: the resulting tree t in $\mathscr{P}_f^\omega 1$ is Barr equivalent to s because $\partial_n t = r^n = \partial_n s$ for every $n < \omega$. Put $\partial_\omega s = t$. We generalize this to all ordinals:

▶ **Definition 5.6.** By the *i-saturation* of a given tree s is meant an i-saturated, strongly extensional tree $\partial_i s$ with $s \sim_i \partial_i s$.

▶ **Example 5.7.** (1) For i finite, Notation 2.4 yields the desired tree.

(2) An example of ω-saturation can be seen in Example 2.10: for the left-hand tree s the second tree is $\partial_\omega s$.

▶ **Remark 5.8.** If t and u are bisimilar trees, then they are equivalent under all of the above equivalences $\sim_i$. This is easy to see by transfinite induction.

Also, if t is i-saturated, then every tree bisimilar to t is i-saturated (in fact, every tree equivalent under $\sim_i$ is). In particular, the *strongly extensional quotient* of a tree t, which is the quotient modulo the largest tree bisimulation $R \subseteq t \times t$, is i-saturated whenever t is.

So even though a tree t is i-saturated it might not be its own i-saturation. Indeed, the third tree in Example 2.10 is ω-saturated but its ω-saturation is the fourth tree.

▶ **Proposition 5.9.** *Every tree has for every ordinal i a unique i-saturation.*

▶ **Remark 5.10.** For infinite ordinals we will see that there is a canonical tree morphism d_i from a tree s to its i-saturation $\partial_i s$. Moreover, if $i = \alpha + n$ where α is a limit ordinal and $n < \omega$, then d_i is surjective when restricted to nodes of depths at most n.

Sketch of proof. (1) Uniqueness. Given i-saturated, strongly extensional trees t and u, we need to prove that $t \sim_i u$ implies $t = u$. The step from i to $i+1$ is easy. For $i = \omega$ this was established in Theorem 2.16. For all other limit ordinals i, this is proved analogously.

(2) Existence. For $i < \omega$ we have $\partial_i s$ as in Section 2. The isolated steps are trivial: given ∂_i we define ∂_{i+1} by taking a tree s and letting s' be the tree-tupling of all $\partial_i s_x$, where x ranges over the children of the root of s. Then the strong extensional quotient of s' is $\partial_{i+1} s$, see Remark 5.8.

The canonical morphism is the composite $d_{i+1} = e \cdot d_i'$ where $d_i' \colon s \to s'$ is the tree morphism acting on every maximal subtree s_x as the corresponding canonical morphism $d_i^{(s_x)} \colon s_x \to \partial_i s_x$, and e is the strong extensional quotient. This composite is, in the case $i = \alpha + n$ for a limit ordinal α, surjective on nodes of depths at most $n+1$ since each $d_i^{(s_x)}$ is surjective on nodes of depths at most n.

For limit ordinals i we construct, for every tree t, the i-saturation in two steps: first t' will be a possibly large tree (with a class of nodes) which is i-saturated and fulfils $t \sim_i t'$. Then $\partial_i t$ is the strong extensional quotient. This is an i-saturation of t because Remark 5.8 holds clearly for large trees, too. And the resulting tree $\partial_i t$ is small, in fact, it is λ_i-branching for a specific cardinal λ_i analogously to $2^{\aleph_0}$ for $i = \omega$ (see Lemma 2.15). ◀

▶ **Theorem 5.11.** *The final chain of $\mathscr{P}$ can be described for all ordinals i by*

$$\mathscr{P}^i 1 = \textit{all } i\textit{-saturated, strongly extensional trees}$$

with the connecting maps into $\mathscr{P}^j 1$ given by ∂_j for all $j < i$.

Sketch of proof. For i finite this is obvious since $\mathscr{P}^i 1 = \mathscr{P}_f^i 1$, for $i = \omega$ use Theorem 2.16.

We proceed by transfinite induction for all infinite ordinals. If the statement holds for i then it holds for $i+1$ provided that every set $M \subseteq \mathscr{P}(\mathscr{P}^i 1) = \mathscr{P}^{i+1} 1$ of trees is identified with the tree-tupling t_M of all members of M. Obviously, t_M is $(i+1)$-saturated and strongly extensional. Conversely, every $(i+1)$-saturated strongly extensional tree is obtained by tree tupling via a unique set M. Thus, $\mathscr{P}^{i+1} 1 = \mathscr{P}(\mathscr{P}^i 1)$ is the set of all $(i+1)$-saturated, strongly extensional trees. If $\partial_j \colon \mathscr{P}^i 1 \to \mathscr{P}^j 1$ is the given connecting map, then the connecting map $\mathscr{P}\partial_j$ corresponds to ∂_{j+1} when the above identification of M and t_M is performed.

Thus, it remains to prove, for every limit ordinal $\alpha > \omega$, that the cone $\partial_i \colon \mathscr{P}^\alpha 1 \to \mathscr{P}^i 1$ $(i < \alpha)$ is a limit cone. This is technically more involved than the proof of Theorem 2.16 but the ideas are similar. ◀

▶ **Remark 5.12.** J. Worrell [21] proved that since $\mathscr{P}_\lambda$ preserves intersections of chains of subobjects and is λ-accessible, the final chain converges in $\lambda + \omega$ steps, where all steps after λ are given by monomorphisms. We can describe the individual steps $\mathscr{P}_\lambda^i 1$ for all $i < \lambda$ in terms of trees. If i is an infinite ordinal then it has the form $i = \alpha + n$, $n < \omega$ and α a limit ordinal.

▶ **Corollary 5.13.** *The final chain of $\mathscr{P}_\lambda$ has the steps $\mathscr{P}_\lambda^i 1$ for all ordinals $i < \lambda$ given by*

$$\mathscr{P}_\lambda^i 1 = \text{all } i\text{-saturated, strongly extensional trees whose } i\text{-saturation is } \lambda\text{-branching at the first } n \text{ levels for all } i = \alpha + n.$$

The connecting maps are $\partial_j \colon \mathscr{P}_\lambda^i 1 \to \mathscr{P}_\lambda^j 1$ for all $j < i$.

▶ **Corollary 5.14.** *Let λ be an infinite cardinal. The final coalgebra for $\mathscr{P}_\lambda$ can be described as the coalgebra of all strongly extensional, λ-branching trees.*

Indeed, each such tree is λ-saturated, see Example 5.4. Thus, it lies in $\mathscr{P}_\lambda^{\lambda+\omega} 1$. Conversely, every tree in $\mathscr{P}_\lambda^{\lambda+\omega} 1$ is λ-branching because the connecting map $\mathscr{P}_\lambda^{\lambda+n+1} 1 \to \mathscr{P}_\lambda^{\lambda+n} 1$ is a monomorphism for every $n < \omega$: this follows from $\mathscr{P}_\lambda$ being a λ-accessible functor, see [21]. Consequently, the limit $\mathscr{P}_\lambda^{\lambda+\omega} 1 = \lim_{n<\omega} \mathscr{P}_\lambda^{\lambda+n} 1$ is the intersection of the subsets $\mathscr{P}_\lambda^{\lambda+n} 1$ of $\mathscr{P}_\lambda^\lambda 1$. Since all tree in $\mathscr{P}_\lambda^{\lambda+n} 1$ are λ-branching at the first n levels, it follows that every tree in $\mathscr{P}_\lambda^{\lambda+\omega} 1$ is λ-branching.

A completely different proof has been presented by D. Schwencke [19].

6 Conclusions and future work

We proved several results which generalize Worrell's description [21] of the final coalgebra of $\mathscr{P}_f$ as the coalgebra of all strongly extensional, finitely branching trees. We described the final coalgebra for the functor $\mathscr{M}_f$ of finite multisets with weights from a given commutative monoid M. This final coalgebra consists of all finitely branching strongly extensional M-labeled trees. This holds for all positive and refinable monoids. Our proof is substantially different from Worrell's, since it is based on congruences on the coalgebra of all extensional trees. We would like to generalize our work on saturated trees to the case of functors $\mathscr{M}_f$. And we plan to apply our methods to probabilistic transition systems.

For $\mathscr{P}_f$ we also described the limit $\mathscr{P}_f^\omega 1$ of the final ω^{op}-chain as the set of all saturated strongly extensional trees, or the set of all maximal consistent theories in the modal logic K. We proved that saturated trees are precisely those trees which are modally saturated in the sense of K. Fine [10]. This also is related to (but quite different from) Worrell's description

of $\mathscr{P}_f^\omega 1$ by means of compactly branching trees. We then generalized saturatedness to i-saturatedness and proved that the final chain $\mathscr{P}^i 1$ of the full power-set functor consists of all strongly extensional i-saturated trees. From this we derived e.g. that the countable power-set functor $\mathscr{P}_c$ has the final coalgebra consisting of all strongly extensional countably branching trees. We leave as open problem the decision whether $\omega_1 + \omega$ is the smallest ordinal for the convergence of the final chain of $\mathscr{P}_c$.

We have characterizations of the initial algebras of the functors $\mathscr{M}_f$. There are open questions concerning the final chains for these functors: for the Boolean monoid, yielding $\mathscr{P}_f$, the chain needs $\omega + \omega$ steps, as proved by Worrell. However, for the monoid $(\mathbb{N}, +, 0)$, the corresponding functor $\mathscr{N}_f$ is analytic, hence, the convergence needs only ω steps. We do not know what happens in the case of a general monoid.

References

1 Abramsky, S.: A cook's tour of the finitary non-wellfounded sets. In S. Artemov et al (eds), *We Will Show Them: Essays in honour of Dov Gabbay*, College Publications, Vol. 1, 1–18, 2005

2 Aczel, P.: *Non-well-founded sets.* Lecture Notes 14, CSLI, Stanford 1988

3 Aczel, P., Mendler, P. F.: A final coalgebra theorem. Lecture Notes in Comput. Sci. **389**, Springer 1989, 357–365

4 Adámek, J.: Free algebras and automata realizations in the language of categories. Comment. Math. Univ. Carolinæ **15** (1974) 589–609

5 Adámek, J., S. Milius, S., Velebil, J.: On coalgebra based on classes. Theoret. Comput. Sci. **316** (2004) 685–705

6 Adámek, J., Trnková, V.: Automata and algebras in categories. Kluwer Academic Publ., Dordrecht 1990

7 Barr, M.: Terminal coalgebras in well-founded set theory Theoret. Comput. Sci. **114** (1993) 299–315

8 Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press 2001

9 Borceux, F.: Handbook of categorical algebra 2. Cambridge University Press 1994

10 Fine, K.: Normal forms in modal logic. Notre Dame J. Formal Logic **16** (1975) 229–237

11 Fine, K.: Some Connections Between Elementary and Modal Logic, in S. Kanger (ed), *Proc. of the Third Scandinavian Logic Symposium.* Amsterdam: North Holland, 1975, 15–31

12 Gumm, H.-P., Schröder T.: Monoid-labelled transition systems. Electron. Notes Theor. Comput. Sci. **44** (2001)

13 Joyal, A.: Foncteurs analytiques et espèce de structure. Lecture Notes in Math. **1234**, Springer 1986, 126–159

14 Kurz, A., Pattinson, D.: Coalgebraic modal logic of finite rank. Math. Structures Comput. Sci. **15:3** (2005), 453–473

15 Lambek, J.: A fixpoint theorem for Complete Categories, Math. Z. **103** (1968), 151–161

16 Moss, L. S.: Finite models constructed from canonical formulas, J. Philos. Logic **36** (2007), 605–640

17 Rutten, J.: A calculus of transition systems, *CSLI lecture notes* **53**, (1995)

18 Rutten, J.: Universal coalgebra: a theory of systems, Theoret. Comput. Sci. **249**, 3–80 (2000)

19 Schwencke, D.: Coequational logic for accessible functors. Inform. and Comput. **208** (2010), 1469–1489

20 Trnková, V.: On descriptive classification of set functors I, Comment. Math. Univ. Carolinæ **12** (1971), 143–174

21 Worrell, J.: On the final sequence of a finitary set functor. Theoret. Comput. Sci. **338** (2005) 184–199

Transfinite Update Procedures for Predicative Systems of Analysis

Federico Aschieri

Dipartimento di Informatica, Università di Torino
Italy
School of EECS, Queen Mary, University of London
United Kingdom

Abstract

We present a simple-to-state, abstract computational problem, whose solution implies the 1-consistency of various systems of predicative Analysis and offers a way of extracting witnesses from classical proofs. In order to state the problem, we formulate the concept of transfinite update procedure, which extends Avigad's notion of update procedure to the transfinite and can be seen as an axiomatization of learning as it implicitly appears in various computational interpretations of predicative Analysis. We give iterative and bar recursive solutions to the problem.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Update procedure, epsilon substitution method, predicative classical analysis, bar recursion

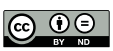
Digital Object Identifier 10.4230/LIPIcs.CSL.2011.20

1 Introduction

The aim of this paper is to provide an abstract description of learning as it is appears in various computational interpretations of predicative fragments of classical second order Arithmetic. Our account has a twofold motivation and interest.

Its first purpose is to provide a foundation that will serve to extend Aschieri and Berardi's learning based realizability for Heyting Arithmetic with EM_1 (see [3]) to predicative fragments of Analysis: a possible path to follow is the one suggested here. In particular, we describe and prove the termination of the learning processes that should arise when extending the approach of learning based realizability to predicative Arithmetic.

Secondly, we continue the work of Avigad on update procedures [4] and extend it to the transfinite case by introducing the concept of transfinite update procedure. The concept may be seen as an axiomatization of learning as implicitly used in the epsilon substitution method for Elementary Analysis and Ramified Analysis as formulated in the work of Mints et al. ([8], [9]). The idea is that one can associate with any classical proof in those systems of any formula $\exists x^{\mathbb{N}} P(x)$, with P computable, a transfinite update procedure. Each transfinite update procedure has a so called *zero*, representing a finite approximation of some transfinite sequence of oracles thanks to which it is possible to compute a n such that $P(n)$ holds. The problem is both to formulate efficient learning processes that build step by step this finite zero and to prove their termination. The notion of update procedure is useful to understand the combinatorial content and the fundamental ideas of the epsilon method in a totally abstract way. By formulating an abstract self-contained concept, we also hope to present to non-specialists a very challenging computational problem, whose efficient solution is of great importance for program extraction purposes in the proof theory of classical logic.



Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 20–34

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

Plan of the Paper. In section §2 we introduce and motivate the concept of transfinite update procedure and give a very short non-constructive proof of the existence of finite zeros.

In section §3 we formulate the notion of "learning process generated by an update procedure" and prove that every learning process terminates with a zero for the associated update procedure. The result represents a semi-constructive proof of the existence of finite zeros and the learning processes generated are "optimal", in the sense that one may provide constructively the best possible ordinal bounds to their length and to the size of finite zeros (by applying techniques of Mints et al. [8]).

In sections §4 and §5 we formalize the notion of update procedure in typed lambda calculus plus bar recursion and prove the existence of zeros for update procedures of ordinal less than ω^ω by writing down simple bar recursive terms. These results are enough to give computational interpretation to proofs of Elementary Analysis and Ramified Analysis, as formulated in [8], [9] (see the full version of this paper [1]). In fact, our methods yield bar recursive proofs of termination for epsilon substitution method as formulated in [9].

Acknowledgments. I'd like to thank Paulo Oliva for his advice throughout this work.

2 Learning in Predicative Analysis

It is very well-known and of fundamental importance that intuitionistic Arithmetic is constructive. This in particular implies that from a proof that there exists an object with a given property, one can always extract a computer program that construct an object with that property. Such feature of intuitionistic Arithmetic depends on the fact that all its axioms and inference rules never assert the existence of something that has not already been implicitly constructed.

In the classical framework, the situation is much different. From the computational point of view, any classical predicative subsystem of second order Arithmetic poses very difficult problems. The axioms that it adds on top of intuitionistic Arithmetic are ontologically very strong. For every formula $\phi(x)$, there is an axiom of comprehension asserting the existence of a function g able to decide the truth of $\phi(x)$:

$$\exists g^{\mathtt{N}\to\mathtt{N}}\forall x^{\mathtt{N}}.\ g(x)=0 \leftrightarrow \phi(x)$$

Axioms of countable choice assert the existence of functions computing existential witnesses of truth of formulas:

$$(\forall x^{\mathtt{N}}\exists y^{A}\ \phi(x,y)) \to \exists g^{\mathtt{N}\to A}\forall x^{\mathtt{N}}\phi(x,g(x))$$

In order to give a *natural* computational interpretation even for the most simple form of the excluded middle

$$\mathsf{EM}_1 : \forall n^{\mathtt{N}}.\exists x^{\mathtt{N}} Pnx \vee \forall y^{\mathtt{N}}\neg Pny$$

one would have to provide a program for deciding, given any number n, the truth of the formula $\exists x^{\mathtt{N}} Pnx$, with P decidable.

Given the situation, a direct computational interpretation of classical logic might seem impossible. Fortunately, there is a fundamental observation that enables us to partially solve this problem: whatever the function or the decision procedure whose existence is assumed, it will be used only a finite number of times in actual computations of finite results! In other words, non-computable functions exist – and we cannot do anything about that – but one only needs to compute finite approximations of them in order to carry out actual computations.

Over this observation, what we call "learning-based computational interpretations" of classical Arithmetic build their success. We include in this category Hilbert's epsilon substitution method, Avigad's update procedures [4] and Aschieri and Berardi's learning-based realizability [3] for intuitionistic Arithmetic plus EM_1.

2.1 Learning Based Realizability for Intuitionistic Arithmetic with EM_1

In the case of learning-based realizability for Intuitionistic Arithmetic with EM_1, one just considers a class of realizers recursive in the oracle for the Halting problem. Such programs easily interpret EM_1, but are ineffective; to recover effectiveness they are evaluated with respect to finite oracle approximations. Since approximations may be inadequate, results of computations may be wrong. But learning-based realizers are *self-correcting* programs, able to identify wrong oracle values used during computations and to correct them with right values that they learn during the same computations. Realizers keep correcting mistakes until they find a good finite approximation of the oracle they use.

2.2 Avigad's Finite Update Procedures for Peano Arithmetic

If one wants to provide a direct computational interpretation of excluded middle EM for arbitrary arithmetical formulas

$$\forall n^{\mathbb{N}}.\ \exists x_1^{\mathbb{N}} \forall y_1^{\mathbb{N}} \dots \exists x_k^{\mathbb{N}} \forall y_k^{\mathbb{N}}\ P(n, x_1, y_1, \dots, x_k, y_k) \vee \forall x_1^{\mathbb{N}} \exists y_1^{\mathbb{N}} \dots \forall x_k^{\mathbb{N}} \exists y_k^{\mathbb{N}} \neg P(n, x_1, y_1, \dots, x_k, y_k)$$

he needs much more computational power: an oracle for the Halting problem is no longer enough. For example, if one wants to interpret

$$\mathsf{EM}_2 := \forall n^{\mathbb{N}}.\ \exists x^{\mathbb{N}} \forall y^{\mathbb{N}} P(n, x, y)\ \vee\ \forall x^{\mathbb{N}} \exists y^{\mathbb{N}} \neg P(n, x, y)$$

he also needs an oracle for the Halting problem for programs *recursive* in the oracle for the Halting problem. This is due to the fact that in order to check, for any fixed m, whether the formula $\forall y^{\mathbb{N}} P(n, m, y)$ is true, one can use a program $p(m)$ recursive in the oracle of the Halting problem. But in order to determine the truth of $\exists x^{\mathbb{N}} \forall y^{\mathbb{N}} P(n, x, y)$, one has to determine whether there exists an m such that $p(m)$ outputs the answer that the formula $\forall y^{\mathbb{N}} P(n, m, y)$ is true.
In general, in order to interpret EM_n one needs a sequence of oracles $\Phi_0, \dots, \Phi_n$ such that for every $1 \leq i \leq n$, Φ_i is an oracle for the Halting problem for programs *recursive* in the subsequence $\{\Phi_j\}_{1 \leq j < i}$. More precisely, let $T_i(x, y, z)$ the predicate, recursive in $\{\Phi_j\}_{1 \leq j < i}$, that holds iff the x-th Kleene's partial recursive function f_x, recursive in $\{\Phi_j\}_{1 \leq j < i}$, terminates on input y after z reduction steps. Then Φ_i must satisfy the following Skolem axiom:

$$\forall x^{\mathbb{N}}, y^{\mathbb{N}}, z^{\mathbb{N}}.\ T_i(x, y, z) \to T_i(x, y, \Phi_i(|x, y|))$$

where $|\_, \_|$ is a bijective coding of pairs of natural numbers into natural numbers.

Now, an Avigad update procedure is a functional $\mathcal{U}$ that takes as argument a finite sequence of functions $f_0, \dots, f_n$ approximating some oracles $\Phi_0, \dots, \Phi_n$. Then, it uses those functions to compute some witnesses for some provable Σ_1^0 formula of Peano Arithmetic PA (i.e. a formula of the form $\exists x^{\mathbb{N}} P(x)$, with P computable). Afterwards, it checks whether the result of its computation is sound. If it is not, it identifies some wrong value of f_i used in the computation and it corrects it with a new one. $\mathcal{U}$, remarkably, can always do that, thanks to the fact that in this case an instance of the Skolem axiom for Φ_i, for some i, (computed by substituting its oracles with their approximation $f_0, \dots, f_n$) must be false. But if an instance

$$T_i(n, m, l) \to T_i(n, m, f_i(|n, m|))$$

is false, then its antecedent is true and its consequent is false. Therefore, $\mathcal{U}$ *learns* a new value for f_i on argument $|n, m|$, namely l, which will replace its former wrong value $f_i(|n, m|)$.

▶ **Definition 1** (Avigad's Finite Update Procedures)**.** A *k-ary update procedure*, $k \in \mathbb{N}^+$, is a continuous function $\mathcal{U} : (\mathbb{N} \to \mathbb{N})^k \to \mathbb{N}^3 \cup \{\emptyset\}$ (i.e., its output is determined by a finite number of values of the input functions) such that the following holds:

1. for all function sequences $f = f_1, \ldots, f_k$

$$\mathcal{U}f = (i, n, m) \implies 1 \leq i \leq k$$

2. for all function sequences $f = f_1, \ldots, f_k$ and $g = g_1, \ldots, g_k$, for all $1 \leq i < k$, if
   - $\mathcal{U}f = (i, n, m)$
   - for all $j < i$, $f_j = g_j$
   - $g_i(n) = m$

   then: $\mathcal{U}g = (i, h, l) \implies h \neq n$.

If $\mathcal{U}$ is a k-ary update procedure, a *zero* for $\mathcal{U}$ is a sequence $f = f_1, \ldots, f_k$ of functions such that $\mathcal{U}f = \emptyset$.

Condition (2) of definition 1 means that the values of the i-th function depend on the values of some of the functions f_j, with $j < i$, and learning on level i is possible only if all the lower levels j have "stabilized". In particular, if $\mathcal{U}$ is a k-ary update procedure and $f : (\mathbb{N} \to \mathbb{N})^k$ is a sequence of functions approximating the oracles $\Phi_1, \ldots, \Phi_k$, there are two possibilities: either f is a fine approximation and then $\mathcal{U}f = \emptyset$; or f is not and then $\mathcal{U}f = (i, n, m)$, for some numerals n, m: $\mathcal{U}$ says the function f_i should be updated as to output m on input n. Moreover, if $\mathcal{U}f = (i, n, m)$, one in a sense has *learned* that $\Phi_i(n) = m$: by definition of update procedure, if g is a function sequence agreeing with f in its first $i - 1$ elements, g_i is another candidate approximation of Φ_i and $g_i(n) = m$, then $\mathcal{U}g$ does not represent a request to modify the value of g_i at point n, for $\mathcal{U}g = (i, h, l)$ implies $h \neq n$.

The main theorem about update procedures is that they always have zeros and these latter can be computed through learning processes *guided* by the former. Intuitively a zero of an update procedure represents a good approximation of the oracles used in a computation, and in particular a good enough one to yield some sought classical witness.

2.3 Transfinite Update Procedures for Predicative Systems of Analysis

In general, learning based computational interpretations of predicative fragments of classical analysis (see Mints et al. [8], [9]) provide answers to the computational challenges of classical axioms by the following three-stage pattern:

1. They identify a sequence $\{\Phi_\beta\}_{\beta<\alpha}$ – possibly transfinite – of non-computable functions $\mathbb{N} \to \mathbb{N}$.
2. They define classical witnesses for provable Σ^0_1 formulas by using programs recursive in Φ.
3. They define update procedures through which it is possible to find, for every particular computation, a suitable finite approximation of the functions of Φ such that one can effectively compute the witnesses defined at stage two.

The functions in the sequence Φ of stage (1) are the computational engine of the interpretation. Given the difficulty of computing witnesses in classical Arithmetic, they are always non-computable. It is therefore not surprising that given this additional computational

power, one is able to define at stage (2) witnesses for classical formulas. If we picture the sequence Φ as a sequence of infinite stacks of numbers, the learning process of point (3) finds a "vertical" approximation of Φ: functions of Φ are infinite stacks of numbers whereas their finite approximations are finite stacks. Moreover, a crucial point is that the sequence Φ is not an arbitrary sequence. In a sense, Φ is also "horizontally" approximated: for every ordinal α, the recursion theoretic Turing degree of Φ_α is approximated by the degrees of Φ_β, for $\beta < \alpha$. This property is very important: in this way, the values of the functions in Φ can be gradually approximated and learned.

More precisely, Φ can be seen as a sequence of functions obtained by transfinite iteration of recursion theoretic *jump* operator (see for example Odifreddi [10]). That is, for every β, if β is a successor, Φ_β has the same Turing degree of an oracle for the halting problem for the class of functions recursive in $\Phi_{\beta-1}$ (jump); if β is limit, Φ_β has the same Turing degre of the function mapping the code of a pair (α, n), with $\alpha < \beta$, into $\Phi_\alpha(n)$ (join or β-jump). A fundamental property of such a sequence is that the assertion $\Phi_\beta(n) = m$ depends only on the values of the functions Φ_α, for $\alpha < \beta$, and the values of Φ_β are learnable in the limit[1] by a program g recursive in the join of Φ_α for $\alpha < \beta$, which is a guarantee that the learning processes will terminate.

We now give an informal example of the kind of analysis which is needed to carry out the first stage of a learning-based interpretation, in the case of Elementary Analysis EA.

▶ **Example 2** (Elementary Analysis)**.** Consider a subsystem EA of second order Peano Arithmetic in which second order quantification is intended to range over arithmetical sets and hence over arithmetical formulas (formulas with only numerical quantifiers and possibly set parameters). Since one has to interpret excluded middle over arbitrary formulas, it is necessary to provide *at least* programs that can decide truth of formulas. Numerical quantifiers correspond to Turing jumps. That is, if we have a program t (with the same function parameters of ϕ) such that for every pair of naturals n, m

$$t(n, m) = \texttt{True} \iff \phi(n, m)$$

then the truth of $\exists x^{\mathbb{N}} \phi(n, x)$ is equivalent to the termination of a program $Q(n)$ exhaustively checking

$$t(n, 0), t(n, 1), t(n, 2), \ldots$$

until it finds - if there exists - an m such that $t(n, m) = \texttt{True}$. Applying the jump operator to the Turing degree t belongs to, one can write down a program χ_t which is able to determine whether $Q(n)$ terminates. That is

$$\chi_t(n) = \texttt{True} \iff \exists x^{\mathbb{N}} \phi(n, x)$$

Similarly, one eliminates universal numerical quantifiers, thanks to the fact that $\forall \equiv \neg\exists\neg$. Iterating these reasoning and applying $2k$ times the jump operator - and given a recursive enumeration $\phi_0, \phi_1, \ldots$, of arithmetical formulas - one can obtain for every Σ^0_{2k} formula

$$\phi_n(m) := \exists x_1^{\mathbb{N}} \forall y_1^{\mathbb{N}} \ldots \exists x_k^{\mathbb{N}} \forall y_k^{\mathbb{N}}\ P(m, x_1, y_1, \ldots, x_k, y_k)$$

a program t_n such that

$$t_n(m) = \texttt{True} \iff \phi_n(m)$$

[1] In the sense of Gold [7]: $\Phi_\beta(n) = m \iff \lim_{k\to\infty} g(n, k) = m$. We remark that we need here the sequence of oracles $\{\Phi_\alpha\}_{\alpha<\beta}$; without it, it would not be to possible learn values of the powerful Φ_β.

Using the ω-jump operator, one can write down a program u such that

$$u(n,m) = \texttt{True} \iff t_n(m) = \texttt{True}$$

and hence

$$u(n,m) = \texttt{True} \iff \phi_n(m)$$

Now a Σ^1_1 formula $\exists f^{\mathtt{N}\to\mathtt{Bool}}\phi_i$ - provided we assume that $f^{\mathtt{N}\to\mathtt{Bool}}$ ranges over arithmetical predicates - can also be expressed as

$$\exists n^{\mathtt{N}} t_i[\lambda m^{\mathtt{N}} u(n,m)/f]$$

So applying again a jump operator to the recursive degree of $t_i[\lambda m^{\mathtt{N}} u(n,m)/f]$, one is able to write a program determining the truth value of $\exists f^{\mathtt{N}\to\mathtt{Bool}}\phi_i$. Iterating this reasoning, one can decide the truth of arbitrary Σ^1_n formulas.
Summing up, in order to decide truth in Elementary Analysis, one needs to apply the jump operator $\omega + \omega$ times and thus produces a sequence Φ of non-computable functions Φ of length $\omega + \omega$. All the programs that we have described can be thought as recursive in some initial segment of Φ.

We are now in a position to introduce our axiomatization of the learning procedures cited in point (3) above.

▶ **Definition 3** (Transfinite Update Procedures)**.** Let $\alpha \geq 1$ be a numerable ordinal. An *update procedure of ordinal* α is a function $\mathcal{U} : (\alpha \to (\mathbb{N} \to \mathbb{N})) \to (\alpha \times \mathbb{N} \times \mathbb{N}) \cup \{\emptyset\}$ such that:

1. $\mathcal{U}$ is continuous. i.e. for any $f : \alpha \to (\mathbb{N} \to \mathbb{N})$ there is a finite subset A of $\alpha \times \mathbb{N}$ such that for every $g : \alpha \to (\mathbb{N} \to \mathbb{N})$ if $f_\gamma(n) = g_\gamma(n)$ for every $(\gamma, n) \in A$, then $\mathcal{U}f = \mathcal{U}g$.

2. For all functions $f, g : \alpha \to (\mathbb{N} \to \mathbb{N})$ and every ordinal $\beta \in \alpha$, if
   - $t(f) = (\beta, n, m)$
   - for all $\gamma < \beta$, $f_\gamma = g_\gamma$
   - $g_\beta(n) = m$

   then: $t(g) = (\beta, i, j) \implies i \neq n$

The concept of transfinite update procedure is a generalization of Avigad's notion of finite update procedure. A transfinite update procedure, instead of taking just a finite number of function arguments, may get as input an arbitrary transfinite sequence of functions, which are intended to approximate a target sequence Φ; as output, it may return an update (β, n, m), which means that the β-th function taken as argument is an inadequate approximation of Φ_β and must be updated as to output m on input n. Condition (2) means that the values for the β-th function depend only on the values of functions of ordinal less than β in the input sequence and an update procedure returns only updates which are *relatively verified* and hence need not to be changed. In this sense, if $\mathcal{U}f = (\beta, n, m)$, one has *learned* that $\Phi_\beta(n) = m$; so if g_β is a candidate approximation of Φ_β and $g_\beta(n) = m$, then $\mathcal{U}g$ does not represent a request to modify the value of g_β at point n, whenever f and g agree on all ordinals less than β.
We remark that the choice of the type for an update procedure is somewhat arbitrary: we could have chosen it to be

$$(\alpha \to (X \to Y)) \to (\alpha \times X \times Y) \cup \{\emptyset\}$$

as long as the elements of the sets X and Y can be coded by finite objects. Since such coding may always be performed by using natural numbers, we choose to consider $X = Y = \mathbb{N}$.

The use of transfinite update procedures made by learning-based computational interpretations of classical arithmetic can be described as follows. Suppose those interpretations are given a provable formula with an attainable computational meaning, for example one of the form $\forall x^{\mathbb{N}} \exists y^{\mathbb{N}}\ Pxy$, with P decidable. Then, for every numeral n, they manage to define a term $t_n : (\alpha \to (\mathbb{N} \to \mathbb{N})) \to \mathbb{N}$ and an update procedure $\mathcal{U}_n$ of ordinal α such that

$$\mathcal{U}_n(f) = \emptyset \implies Pn(t_n(f))$$

for all $f : \alpha \to (\mathbb{N} \to \mathbb{N})$. The idea is that a witness for the formula $\exists y^{\mathbb{N}} Pny$ is calculated by t_n with respect to a particular approximation f of the oracle sequence Φ we have previously described. If the formula $Pn(t_n(f))$ is true, there is nothing to be done. If it is false, then $\mathcal{U}_n(f) = (\beta, n, m)$ for some β, n, m: a new value for Φ_β is learned. This is what we call "learning by counterexamples": from every failure a new positive fact is acquired. An instance of this kind of learning appears in the case on learning-based realizability for $\mathsf{HA} + \mathsf{EM}_1$, when one defines realizability for atomic formulas: in that case the pair (n, m) is produced by the realizer of the excluded middle. We have seen another example in the section on Avigad update procedures for PA and will see a further one in the full version of this paper ([1]) in the case $\alpha = \omega + k$, with $k \in \mathbb{N}$: the triple (β, n, m) will be produced through the evaluation of Skolem axioms for epsilon terms in the system EA. In general, the Skolem axioms used to define oracles are those who make possible learning by counterexamples.

The effectiveness of the above approach depends on the fact that every update procedure has a finite zero, as defined below.

▶ **Definition 4** (Finite Functions, Finite Zeros, Truncation and Concatenation of Function Sequences)**.** Let $\mathcal{U}$ be an update procedure of ordinal α.

1. $f : \alpha \to (\mathbb{N} \to \mathbb{N})$ is said to be a *finite function* if the set of (γ, n) such that $f_\gamma n \neq 0$ is finite.
2. A *finite zero* for $\mathcal{U}$ is a finite function $f : \alpha \to (\mathbb{N} \to \mathbb{N})$ such that $\mathcal{U}f = \emptyset$.
3. Let $f : \alpha \to (\mathbb{N} \to \mathbb{N})$ and $\beta < \alpha$. Let $f_{<\beta} : \beta \to (\mathbb{N} \to \mathbb{N})$ be the *truncation* of f at β:
$$f_{<\beta} := \gamma \in \beta \mapsto f_\gamma$$
4. Let α_1, α_2 be two ordinals, $f_1 : \alpha_1 \to (\mathbb{N} \to \mathbb{N})$ and $f_2 : \alpha_2 \to (\mathbb{N} \to \mathbb{N})$. Then the *concatenation* $f_1 * f_2 : (\alpha_1 + \alpha_2) \to (\mathbb{N} \to \mathbb{N})$ of f and g is defined as:
$$(f * g)_\gamma(n) := \begin{cases} f_\gamma(n) & \text{if } \gamma < \alpha_1 \\ g_\beta(n) & \text{if } \gamma = \alpha_1 + \beta < \alpha_1 + \alpha_2 \end{cases}$$
5. With a slight abuse of notation, a function $f : \mathbb{N} \to \mathbb{N}$ will be sometimes identified with the corresponding length-one sequence of functions $0 \mapsto (n \in \mathbb{N} \mapsto f(n))$.

We now prove that every update procedure has a finite zero. We will give other more and more constructive proofs of this theorem, that will allow to compute finite zeros for update procedures and thus witnesses for classically provable formulas, thanks to learning-based interpretations. But for now we are only interested in understanding the reason of the theorem's *truth* and give a very short non-constructive proof. All the subsequent proofs can be seen as more and more sophisticated and refined constructivizations of the following argument.

▶ **Theorem 5** (Zero Theorem for Update Procedures of Ordinal α). *Let $\mathcal{U}$ be an update procedure of ordinal α. Then $\mathcal{U}$ has a finite zero.*

Proof. We define, by transfinite induction, a function $f : \alpha \to (\mathbb{N} \to \mathbb{N})$ as follows. Suppose we have defined $f_\gamma : \mathbb{N} \to \mathbb{N}$, for every $\gamma < \beta$. Define the sequence $f_{<\beta} : \beta \to (\mathbb{N} \to \mathbb{N})$ of them all

$$f_{<\beta} := \gamma \in \beta \mapsto f_\gamma$$

Then define

$$f_\beta(x) = \begin{cases} 0 & \text{if } \forall g^{(\alpha-\beta)\to(\mathbb{N}\to\mathbb{N})}\ \forall z^{\mathbb{N}}\ \mathcal{U}(f_{<\beta} * g) \neq \langle \beta, x, z\rangle \\ y & \text{otherwise, for some } y \text{ such that } \exists g^{(\alpha-\beta)\to(\mathbb{N}\to\mathbb{N})}\ \mathcal{U}(f_{<\beta} * g) = \langle \beta, x, y\rangle \end{cases}$$

By axiom of choice and classical logic, for every β, $f_{<\beta}$ and f_β are well defined. So we can let

$$f := f_{<\alpha}$$

Suppose $\mathcal{U}(f) = \langle \beta, x, z\rangle$, for some $\beta < \alpha$: we show that it is impossible. For some $h : (\alpha - (\beta + 1)) \to (\mathbb{N} \to \mathbb{N})$, $f = f_{<\beta} * f_\beta * h$. Hence, for some $g : (\alpha - \beta) \to (\mathbb{N} \to \mathbb{N})$

$$\mathcal{U}(f_{<\beta} * g) = \langle \beta, x, y\rangle \wedge f_\beta(x) = y$$

by definition of f. But $\mathcal{U}$ is an update procedure and so

$$(\mathcal{U}(f_{<\beta} * g) = \langle \beta, x, y\rangle \wedge f_\beta(x) = y \wedge \mathcal{U}(f_{<\beta} * f_\beta * h) = \langle \beta, x, z\rangle) \implies x \neq x$$

which is impossible. We conclude that $\mathcal{U}(f) = \emptyset$ and, by continuity, that $\mathcal{U}$ has a finite zero. ◀

3 Learning Processes Generated by Transfinite Update Procedures

In this section we show that every update procedure $\mathcal{U}$ generates a learning process and this learning process always terminates with a finite zero of $\mathcal{U}$. This result is an abstract version of the termination of the H-process as defined in the various versions of epsilon substitution method (see Mints et al. [8]). The proof of termination is semi-constructive and is similar to the one in Mints et al. [8] (which however is by contradiction while ours is not).

If $\mathcal{U}$ is an update procedure and $\mathcal{U}(f) = \langle \gamma, n, m\rangle$, then the value of f_γ at argument n must be updated as to be equal to m. But as explained in the introduction, we may imagine that all the values of all the functions f_β, with $\beta > \gamma$, depend on the values of the *current* f_γ. Therefore, if we change some of the values of f_γ, we must erase all the values of all the functions f_β, for $\beta > \gamma$, because they may be inconsistent with the new values of f_β. In a sense, f is a fragile structure, that may be likened to an *house of cards*: if we change something in a layer, then all the higher ones collapse. We define an update operator $\oplus$ that performs those operations.

▶ **Definition 6** (Controlled Update of Functions). Let $f : \alpha \to (\mathbb{N} \to \mathbb{N})$ and $\langle \gamma, n, m\rangle \in \alpha \times \mathbb{N} \times \mathbb{N}$. We define a function $f \oplus \langle \gamma, n, m\rangle : \alpha \to (\mathbb{N} \to \mathbb{N})$ such that

$$(f \oplus \langle \gamma, n, m\rangle)_\beta(x) := \begin{cases} f_\beta(x) & \text{if } \beta < \gamma \text{ or } (\beta = \gamma \text{ and } x \neq n) \\ m & \text{if } \gamma = \beta \text{ and } x = n \\ 0 & \text{otherwise} \end{cases}$$

We also define $f \oplus \emptyset := f$.

We now define the concept of "learning process generated by an update procedure $\mathcal{U}$". It may be thought as a process of updating and learning new values of functions, which is *guided* by $\mathcal{U}$. It corresponds to the step three of the learning based computational interpretations of classical arithmetic we have described in the introduction. Intuitively, such a learning process starts from the always zero function 0^α. If $\mathcal{U}$ says that some value of 0^α must be updated - i.e. $\mathcal{U}(0^\alpha) = \langle \gamma, n, m\rangle$ - then the learning process generates the function $\mathcal{U}^{(1)} := 0^\alpha \oplus \langle \gamma, n, m\rangle$. Similarly, if $\mathcal{U}$ says that some value of $\mathcal{U}^{(1)}$ must be updated - i.e. $\mathcal{U}(\mathcal{U}^{(1)}) = \langle \gamma', n', m'\rangle$ - then the learning process generates the function $\mathcal{U}^{(2)} := \mathcal{U}^{(1)} \oplus \langle \gamma', n', m'\rangle$. The process goes on indefinitely in the same fashion.

▶ **Definition 7** (Learning Processes Generated by $\mathcal{U}$)**.** Let $\mathcal{U}$ be an update procedure of ordinal α. For every $n \in \mathbb{N}$, we define a function $\mathcal{U}^{(n)} : \alpha \to (\mathbb{N} \to \mathbb{N})$ by induction as follows:

$$\mathcal{U}^{(0)} := 0^\alpha := \gamma \in \alpha \mapsto (n \in \mathbb{N} \mapsto 0)$$
$$\mathcal{U}^{(n+1)} := \mathcal{U}^{(n)} \oplus \mathcal{U}(\mathcal{U}^{(n)})$$

Moreover, a function $f : \alpha \to (\mathbb{N} \to \mathbb{N})$ is said to be $\mathcal{U}$-*generated* if there exists an n such that $f = \mathcal{U}^{(n)}$.

The aim of the learning process generated by $\mathcal{U}$ is to find a finite zero for $\mathcal{U}$. Indeed, if for some n, $\mathcal{U}(\mathcal{U}^{(n)}) = \emptyset$, then for all $m \geq n$, $\mathcal{U}^{(m)} = \mathcal{U}^{(n)}$ and we thus say that the learning process *terminates*. We now devote ourselves to the proof that learning processes always terminate. In other words, every update procedure $\mathcal{U}$ has a $\mathcal{U}$-generated finite zero.

Given an update procedure $\mathcal{U}$, its useful to define a new "simpler" update procedure, obtained from $\mathcal{U}$ by fixing some initial segment of its input, ignoring all updates relative to this fixed part of the input and adjusting their indices.

▶ **Definition 8.** Let $\mathcal{U}$ be an update procedure of ordinal α. Then, for any function $g : \beta \to (\mathbb{N} \to \mathbb{N})$, with $\beta < \alpha$, define a function

$$\mathcal{U}_g : ((\alpha - \beta) \to (\mathbb{N} \to \mathbb{N})) \to (\alpha - \beta) \times \mathbb{N} \times \mathbb{N} \cup \{\emptyset\}$$

as follows:

$$\mathcal{U}_g(f) = \begin{cases} \langle \gamma, n, m\rangle & \text{if } \mathcal{U}(g * f) = \langle \beta + \gamma, n, m\rangle \\ \emptyset & \text{otherwise} \end{cases}$$

(We point out that if $\beta = 0 = \emptyset$, $\mathcal{U}_g = \mathcal{U}$ as it should be)

Indeed $\mathcal{U}_g$ as defined above is an update procedure.

▶ **Lemma 9.** *Let $\mathcal{U}$ be an update procedure of ordinal α. Then, for any function $g : \beta \to (\mathbb{N} \to \mathbb{N})$, with $\beta < \alpha$:*

1. *$\mathcal{U}_g$ is an update procedure of ordinal $\alpha - \beta$.*
2. *For every $h : \mathbb{N} \to \mathbb{N}$, $\mathcal{U}_{g*h} = (\mathcal{U}_g)_h$.*

Proof. Immediate. ◀

The strategy of our termination proof can be described as follows. Given an update procedure $\mathcal{U}$ of ordinal α, we shall define a sequence of functions $g : \alpha \to (\mathbb{N} \to \mathbb{N})$ such that a "reduction lemma" can be proved: if, for some $\beta < \alpha$, $\mathcal{U}_{g_{<\beta}}$ has a $\mathcal{U}_{g_{<\beta}}$-generated finite zero, then for some $\gamma < \beta$ also $\mathcal{U}_{g_{<\gamma}}$ has a $\mathcal{U}_{g_{<\gamma}}$-generated finite zero. But the greater the ordinal β the easier it is to compute with a learning process a finite zero for $\mathcal{U}_{g_{<\beta}}$, because

the sequence $g_{<\beta}$ becomes so long that the input for $\mathcal{U}_{<\beta}$ becomes short. So we shall be able to show that for some large enough β, $\beta < \alpha$, $\mathcal{U}_{g_{<\beta}}$ has a $\mathcal{U}_{g_{<\beta}}$-generated finite zero, which proves the theorem in combination with the reduction lemma. This technique can be seen as a generalization of Avigad's [4] to the transfinite case.

We now prove the reduction lemma in the limit case.

▶ **Lemma 10** (Reduction Lemma, Limit Case). *Let $\mathcal{U}$ be an update procedure of ordinal α and $g : \beta \to (\mathbb{N} \to \mathbb{N})$, with β limit ordinal and $\beta < \alpha$. Then*

1. *If $f : (\alpha - \beta) \to (\mathbb{N} \to \mathbb{N})$ is $\mathcal{U}_g$-generated, then there exists $\gamma < \beta$ such that $0^{\beta-\gamma} * f$ is $\mathcal{U}_{g_{<\gamma}}$-generated.*

2. *If $\mathcal{U}_g$ has a $\mathcal{U}_g$-generated finite zero, then there exists $\gamma < \beta$ such that $\mathcal{U}_{g_{<\gamma}}$ has a $\mathcal{U}_{g_{<\gamma}}$-generated finite zero.*

Proof. See the full version of this paper [1]. ◀

We now prove the reduction lemma in the successor case.

▶ **Lemma 11** (Reduction Lemma, Successor Case). *Let $\mathcal{U}$ be an update procedure of ordinal α. Define $g : \mathbb{N} \to \mathbb{N}$ as follows:*

$$g(x) := \begin{cases} y & \text{if } \exists i.\ \mathcal{U}(\mathcal{U}^{(i)}) = \langle 0, x, y\rangle \wedge i = \min\{n \mid \exists z\ \mathcal{U}(\mathcal{U}^{(n)}) = \langle 0, x, z\rangle\} \\ 0 & \text{otherwise} \end{cases}$$

Then:

1. *For every finite function $g_0 \leq {}^2 g$, if $g_0 * 0^{\alpha-1}$ is $\mathcal{U}$-generated and f is $\mathcal{U}_{g_0}$-generated, then $g_0 * f$ is $\mathcal{U}$-generated.*

2. *If $\mathcal{U}_g$ has a $\mathcal{U}_g$-generated finite zero, then $\mathcal{U}$ has a $\mathcal{U}$-generated finite zero.*

Proof. See the full version of this paper [1]. ◀

We are now able to prove the main theorem: update procedures generate terminating learning processes.

▶ **Theorem 12** (Termination of Learning Processes). *Let $\mathcal{U}$ be an update procedure of ordinal α. Then, $\mathcal{U}$ has a finite zero. In particular, there exists $k \in \mathbb{N}$ such that $\mathcal{U}(\mathcal{U}^{(k)}) = \emptyset$.*

Proof. See the full version of this paper [1]. ◀

4 Spector's System B and Typed Update Procedures of Ordinal ω^k

Zeros of trasfinite update procedures cannot in general be computed in Gödel's system T: as we will show, already update procedures of ordinal $\omega + k$, with $k \in \omega$, can be used to give computational interpretation to Elementary Analysis and hence their zeros can be used to compute the functions provably total in Elementary Analysis. We will show however that Spector's system B is enough to compute zeros.

[2] We define $g_0 \leq g$ iff for all x $g_0(x) \neq 0 \implies g_0(x) = g(x)$

▶ **Definition 13** (Bar Recursion Operator, Spector's System B, Type Level of Bar Recursion)**.** In the following, we will work with Spector's system B (see Spector [12] and, for a modern exposition, Kohlenbach [6]) which is Gödel's T augmented with constants $\mathsf{BR}_{\tau,\sigma}, \Psi_{\tau,\sigma}$ respectively of type

$$T_1 \to T_2 \to T_3 \to T_4 \to \tau \qquad \text{and} \qquad T_1 \to T_2 \to T_3 \to T_4 \to \mathtt{Bool} \to \tau$$

with

$$T_1 = (\mathtt{N} \to \sigma) \to \mathtt{N}$$

$$T_2 = \sigma^* \to \tau$$

$$T_3 = \sigma^* \to (\sigma \to \tau) \to \tau$$

$$T_4 = \sigma^*$$

where σ^* is a type representing finite sequences of objects of type σ. The meaning of $\mathsf{BR}_{\tau,\sigma}$ is defined by the equation

$$\mathsf{BR}_{\tau,\sigma} YGHs \stackrel{\tau}{=} \begin{cases} Gs & \text{if } Y\hat{s} < |s| \\ Hs(\lambda x^{\sigma} \mathsf{BR}_{\tau,\sigma} YGH(s * x)) & \text{otherwise} \end{cases} \tag{1}$$

where $s * x$ denotes the finite sequence s followed by x, $\hat{s}$ denotes the function mapping n to s_n, if $n < |s|$, to 0^{σ} otherwise, where s_n is the n-th element of s and $|s|$ is the number of elements in s. If σ, τ, Y, G, H are determined by the context, we we will just write $\mathsf{BR}(s)$ in place of $\mathsf{BR}_{\tau,\sigma} YGHs$.
$\mathsf{BR}_{\tau,\sigma}$ is said to be *bar recursion of type* σ. The *type level* of bar recursion $\mathsf{BR}_{\tau,\mathtt{N}}$ of type $\mathtt{N}$ (said also type 0), is the type level of the constant $\mathsf{BR}_{\tau,\mathtt{N}}$, that is, assuming $\mathtt{N}^* = \mathtt{N}$, $\mathsf{max}(1, \mathsf{typelevel}(\tau)) + 2$.
In order to obtain a strongly normalizing system such that equation 1 holds, we have to add to system B the following reduction rules (for a proof of strong normalization, see Berger [5]):

$$\mathsf{BR}_{\tau,\sigma} YGHs \mapsto \Psi_{\tau,\sigma} YGHs(Y\hat{s} < |s|)$$
$$\Psi_{\tau,\sigma} YGHs(\mathtt{True}) \mapsto Gs$$
$$\Psi_{\tau,\sigma} YGHs(\mathtt{False}) \mapsto Hs(\lambda x^{\sigma} \mathsf{BR}_{\tau,\sigma} YGH(s * x))$$

where $<$ is a term coding the correspondent relation on natural numbers.

Since we are interested only in computable update procedures, we now fix a system for representing them. For the aim of computationally interpreting Elementary Analysis, update procedures can be assumed to belong to system T. However, for more powerful systems one may need more capable update procedures, so we define them to belong to B. Here, we limit ourselves to the ordinal ω^k, for $k \in \omega$, since this ordinal is enough to interpret Elementary Analysis and even fragments of Ramified Analysis (see for example, Mints et al. [9])

▶ **Definition 14** (Representation of Ordinals and Typed Update Procedures of Ordinal ω^k)**.** We will represent ordinal numbers of the form ω^k, with $k \in \omega$, by exploiting the order isomorphism between ω^k and $\mathbb{N}^k$ lexicographically ordered. So, for $k \in \omega$, $k > 0$, we set

$$[\omega^0] := \nu, [\omega^k] := \mathtt{N}^k$$

where ν is the empty string and

$$[\omega^0 \to (\mathbb{N} \to \mathbb{N})] := \mathtt{N} \to \mathtt{N}$$

and, if $k \in \omega$

$$[\omega^{k+1} \to (\mathbb{N} \to \mathbb{N})] := \mathtt{N} \to [\omega^k \to (\mathbb{N} \to \mathbb{N})]$$

where $\mathtt{N}$ is the type representing $\mathbb{N}$ in typed lambda calculus. Define moreover

$$[(\omega^k \times \mathbb{N} \times \mathbb{N}) \cup \{\emptyset\}] := [\omega^k] \times \mathtt{N} \times \mathtt{N}$$

Unfortunately, $\emptyset$ does not have a code. So we have to use an injective coding $|\_|$ of the set $(\omega^k \times \mathbb{N} \times \mathbb{N}) \cup \{\emptyset\}$ into the set of closed normal terms of type $[(\omega^k \times \mathbb{N} \times \mathbb{N}) \cup \{\emptyset\}]$. To fix ideas, we define $|(\beta, n, m)| = \langle \beta', n+1, m+1 \rangle$, with $\beta' : \mathtt{N}^k$ the code of β, and $|\emptyset| = \langle \langle 0, \ldots, 0 \rangle, 0, 0 \rangle$. A *typed update procedure of ordinal* ω^k is a term of Spector's system B of type:

$$[\omega^k \to (\mathbb{N} \to \mathbb{N})] \to [(\omega^k \times \mathbb{N} \times \mathbb{N}) \cup \{\emptyset\}]$$

satisfying point (2) of definition 3, where for simplicity function quantification is assumed to range over functions definable in system B. Equality as it appears in the definition is supposed to be extensional.

5 Bar Recursion Proof of the Zero Theorem for Typed Update Procedures of Ordinal ω^k

In this section we give a constructive proof of the Zero theorem for typed update procedures of ordinal less than ω^k. In particular we show that finite zeros can be computed with bar recursion of type 1. We start with the base case.

▶ **Theorem 15** (Zero Theorem for Update Procedures of Ordinal $1{=}\omega^0$)**.** *Let $\mathcal{U}$ be a typed update procedure of ordinal 1. Then $\mathcal{U}$ has a finite zero σ. Moreover, σ can be calculated as the normal form of a bar recursive term $\mathsf{Zero}(\mathcal{U})$ (defined uniformly on the parameter $\mathcal{U}$) of system T plus bar recursion of type 0.*

The result follows by Oliva [11]. We give below another proof, which is a simplification of Oliva's one, made possible by the slightly stronger condition we have imposed on the notion of update procedure.
The informal idea of the construction - but with some missing justifications - is the following. We reason over the well-founded tree of finite sequences of numbers s such that $\mathcal{U}(\hat{s}) = |(n, m)|$ and $n \geq |s|$. We want to construct a function $\sigma : \mathtt{N} \to \mathtt{N}$ which is a zero of $\mathcal{U}$. Suppose that we have constructed a "good" initial approximation $\sigma(0) * \cdots * \sigma(i)$ of σ; we want to prove that it can be extended to a long enough approximation of σ. Our first step is to continue with $\sigma(0) * \cdots * \sigma(i) * 0$. If this is a good guess, by well-founded induction hypothesis, we can extend $\sigma(0) * \cdots * \sigma(i) * 0$ to a complete approximation $\sigma(0) * \cdots * \sigma(n)$ of σ, with $n > i$. Since we are not sure that our previous guess was lucky, we compute $\mathcal{U}(\sigma(0) * \cdots * \sigma(n))$. If for all m

$$\mathcal{U}(\sigma(0) * \cdots * \sigma(n)) \neq |(i+1, m)|$$

then our approximation for $\sigma(i+1)$ is adequate, and we claim that $\sigma(0) * \cdots * \sigma(n)$ is the approximation of σ we were seeking. Otherwise

$$\mathcal{U}(\sigma(0) * \cdots * \sigma(n)) = |(i+1, m)|$$

for some m: $\mathcal{U}$ tells us that our guess for the value of $\sigma(i+1)$ is wrong. But now we know that $\sigma(0) * \cdots * \sigma(i) * m$ is a good initial approximation of σ and we have made progress. Again by well-founded induction hypothesis, we conclude that we can extend $\sigma(0) * \cdots * \sigma(i) * m$ to a good approximation of σ.

Proof. *of Theorem* 15. We formalize and complete the previous informal argument. In the following s will be a variable for finite sequences of numbers. Using bar recursion of type 0, we can define a term which builds directly the finite zero we are looking for and is such that:

$$\mathsf{BR}(s) = \begin{cases} \hat{s} & \text{if } \mathcal{U}\hat{s} = |(n,m)| \text{ and } n < |s| \\ \hat{s} & \text{if } \mathcal{U}\hat{s} = |\emptyset| \\ \mathsf{BR}(s * m) & \text{if } \mathcal{U}(\mathsf{BR}(s * 0)) = |(|s|, m)| \\ \mathsf{BR}(s * 0) & \text{if } \mathcal{U}(\mathsf{BR}(s * 0)) \neq |(|s|, m)| \text{ for all } m \end{cases}$$

(we assume that $\mathsf{BR}(s)$ checks in order every condition in its definition and executes the action corresponding to the first statisfied condition). We let σ be the normal form of

$$\mathsf{Zero}(\mathcal{U}) := \mathsf{BR}(\langle\rangle)$$

where $\langle\rangle$ is the empty sequence. Let us prove that σ is a finite zero of $\mathcal{U}$. Suppose $\mathcal{U}\sigma = |(n,m)|$: by showing that this is impossible, we obtain that $\mathcal{U}\sigma = |\emptyset|$. The normalization of $\mathsf{BR}(\langle\rangle)$ leads to the following chain of equations:

$$\begin{aligned} \mathsf{BR}(\langle\rangle) &= \mathsf{BR}(\sigma(0)) \\ &= \mathsf{BR}(\sigma(0) * \sigma(1)) \\ &\dots \\ &\dots \\ &= \mathsf{BR}(\sigma(0) * \dots * \sigma(i)) \\ &= \sigma(0)\widehat{\ *\cdots*\ }\sigma(i) \\ &= \sigma \end{aligned}$$

with

$$n < |\sigma(0) * \dots * \sigma(i)| = i + 1$$

In particular $\mathsf{BR}(\langle\rangle) = \mathsf{BR}(\sigma(0) * \dots * \sigma(n-1))$. Now, we have two cases:

1. $\mathcal{U}(\mathsf{BR}(\sigma(0) * \dots * \sigma(n-1) * 0)) = |(n,l)|$. Then

   $$\mathsf{BR}(\langle\rangle) = \mathsf{BR}(\sigma(0) * \dots * \sigma(n-1) * l)$$

   and so $\sigma(n) = l$, which is impossible, by definition 3 of update procedure, point (2), for $\mathcal{U}\sigma = |(n,m)|$.

2. for all l, $\mathcal{U}(\mathsf{BR}(\sigma(0) * \dots * \sigma(n-1) * 0)) \neq |(n,l)|$. Then by definition

   $$\mathsf{BR}(\sigma(0) * \dots * \sigma(n-1)) = \mathsf{BR}(\sigma(0) * \dots * \sigma(n-1) * 0)$$

   Therefore

   $$|(n,m)| = \mathcal{U}\sigma = \mathcal{U}(\mathsf{BR}(\langle\rangle)) = \mathcal{U}(\mathsf{BR}(\sigma(0) * \dots * \sigma(n-1) * 0))$$

   again impossible, by assumption of this case.

We have then proved that σ is the sought finite zero. ◀

We now prove that every typed update procedure of ordinal ω has a finite zero.

▶ **Theorem 16** (Zero Theorem for Typed Update Procedures of Ordinal ω)**.** *Let $\mathcal{U}$ be a typed update procedure of ordinal ω. Then $\mathcal{U}$ has a finite zero σ. Moreover, σ can be calculated as the normal form of a bar recursive term $\mathsf{Zero}_\omega(\mathcal{U})$ (defined uniformly on the parameter $\mathcal{U}$) of system T plus bar recursion of type $1 := \mathtt{N} \to \mathtt{N}$.*

Proof. The finite function $\sigma : [\omega \to (\mathbb{N} \to \mathbb{N})]$ we are going to construct can be represented as a finite function sequence $\sigma(0) * \sigma(1) * \cdots * \sigma(n)$, for a large enough n. In the following s is a variable ranging over finite sequences of natural number functions. Using bar recursion of type 1, we can define in a most simple way a term which builds directly the finite zero we are looking for. We present the construction gradually. To begin with, suppose we are able to define - uniformly on s - terms $\mathsf{BR}(s)$ and $g_s : (\mathtt{N} \to \mathtt{N})$ satisfying the following equation for every s:

$$\mathsf{BR}(s) = \begin{cases} \hat{s} & \text{if } \mathcal{U}\hat{s} = |(\gamma, n, m)| \text{ and } \gamma < |s| \\ \hat{s} & \text{if } \mathcal{U}\hat{s} = |\emptyset| \\ \mathsf{BR}(s * g_s) & \text{otherwise, where } \forall n, m \;\; \mathcal{U}(\mathsf{BR}(s * g_s)) \neq (|s|, n, m) \end{cases}$$

Let

$$\sigma := \mathsf{Zero}_\omega(\mathcal{U}) := \mathsf{BR}(\langle\rangle)$$

We prove that σ is a finite zero of $\mathcal{U}$. We show this by proving that $\mathcal{U}\sigma = (\gamma, n, m)$ is impossible. As in the proof of theorem 15

$$\mathsf{BR}(\langle\rangle) = \mathsf{BR}(\sigma(0) * \cdots * \sigma(i)) = \sigma(0) \widehat{* \cdots *} \sigma(i)$$

with $\gamma < i + 1$. Let

$$r := \sigma(0) * \cdots * \sigma(\gamma - 1)$$

By some computation

$$\begin{aligned} \mathcal{U}\sigma &= \mathcal{U}(\mathsf{BR}(\langle\rangle)) \\ &= \mathcal{U}(\mathsf{BR}(\sigma(0) * \cdots * \sigma(\gamma - 1))) \\ &= \mathcal{U}(\mathsf{BR}(r)) \\ &= \mathcal{U}(\mathsf{BR}(r * g_r)) \end{aligned}$$

Since by construction for all n, m

$$\mathcal{U}(\mathsf{BR}(r * g_r)) \neq |(|r|, n, m)| = |(\gamma, n, m)|$$

we obtain that $\mathcal{U}\sigma \neq (\gamma, n, m)$: impossible.
It remains to show that a g_s such that appears in the definition of $\mathsf{BR}(s)$ exists. Indeed, it is enough to set

$$g_s := \mathsf{Zero}(\lambda f^{\mathtt{N}\to\mathtt{N}} \mathcal{U}_{|s|}(\mathsf{BR}(s * f)))$$

where, for $i \in \mathbb{N}$, we have defined

$$\mathcal{U}_i := \lambda f^{\mathtt{N}\to(\mathtt{N}\to\mathtt{N})}. \text{ if } \mathcal{U}(f) = |(i, n, m)| \text{ then } |(n, m)| \text{ else } |\emptyset|$$

We prove now that in fact $\mathcal{U}(\mathsf{BR}(s * g_s)) \neq |(|s|, n, m)|$ for all n, m. First, observe again that for every s

$$\mathsf{BR}(s) = s * h_1 \widehat{* \cdots *} h_n$$

for some terms $h_1, \ldots, h_n$ of type $\mathtt{N} \to \mathtt{N}$. Now, fix any finite sequence s of type-$\mathtt{N} \to \mathtt{N}$ terms. We want to show that

$$F_s := \lambda f^{\mathtt{N}\to\mathtt{N}} \mathcal{U}_{|s|}(\mathsf{BR}(s * f))$$

is an update procedure of ordinal 1. Suppose $F_s g_1 = |(n, m)|$, $g_2(n) = m$ and $F_s g_2 = |(h, l)|$. Then, by definition of F_s, it must be that

$$\mathcal{U}(\mathsf{BR}(s * g_1)) = |(|s|, n, m)| \qquad \text{and} \qquad \mathcal{U}(\mathsf{BR}(s * g_2)) = |(|s|, h, l)|$$

Moreover,

$$\mathsf{BR}(s * g_2)_{|s|}(n) = g_2(n) = m$$

Since $\mathcal{U}$ is an update procedure, $h \neq n$ must hold; therefore F_s is an update procedure of ordinal 1. But by definition of g_s, Zero and theorem 15, this means that

$$|\emptyset| = F_s(\mathsf{Zero}(F_s)) = \mathcal{U}_{|s|}(\mathsf{BR}(s * g_s))$$

By definition of $\mathcal{U}_{|s|}$ it must be true that $\mathcal{U}(\mathsf{BR}(s * g_s)) \neq |(|s|, n, m)|$ for all n, m. ◀

The previous argument can be generalized in order to prove the Zero theorem for typed update procedures of ordinal ω^k.

▶ **Theorem 17** (Zero Theorem for Typed Update Procedures of Ordinal ω^k, with $k \in \omega$)**.** *Let $\mathcal{U}$ be a typed update procedure of ordinal ω^k. Then $\mathcal{U}$ has a finite zero σ. Moreover, σ can be calculated as the normal form of a bar recursive term $\mathsf{Zero}_{\omega^k}(\mathcal{U})$ (defined uniformly on the parameter $\mathcal{U}$) of system T plus bar recursion of some type A, where $\mathsf{typelevel}(A) = 1$.*

Proof. The idea is to break the zero we want to construct in ω blocks of length ω^{k-1} and build each block by using Zero_{ω^k}. See the full version of this paper [1]. ◀

References

1 F. Aschieri, *Learning in Predicative Analysis*, chapter 6 of [2]

2 F. Aschieri, *Learning, Realizability and Games in Classical Arithmetic*, PhD Thesis, 2011 http://arxiv.org/abs/1012.4992

3 F. Aschieri, S. Berardi, *Interactive Learning-Based Realizability for Heyting Arithmetic with* EM_1, Logical Methods in Computer Science, 2010

4 J. Avigad, *Update Procedures and 1-Consistency of Arithmetic*, Mathematical Logic Quarterly, vol. 48, 2002

5 U. Berger, *Continuous Semantics for Strong Normalization*, LNCS 3526, pag. 23-34, 2005

6 U. Kohlenbach, *Applied Proof Theory*, Springer-Verlag, Berlin, Heidelberg, 2008

7 E. M. Gold, *Limiting Recursion*, Journal of Symbolic Logic 30, pag. 28-48, 1965

8 G. Mints, S. Tupailo, W. Bucholz, *Epsilon Substitution Method for Elementary Analysis*, Archive for Mathematical Logic, volume 35, 1996

9 G. Mints, S. Tupailo, *Epsilon Substitution Method for the Ramified Language and* Δ^1_1*-Comprehension Rule*, Logic and Foundations of Mathematics, 1999

10 P. Odifreddi, *Classical Recursion Theory*, Studies in Logic and Foundations of Mathematics, Elsevier, 1989

11 P. Oliva, *Understanding and Using Spector's Bar Recursive Interpretation of Classical Analysis*, Proceedings of CiE'2006, LNCS, vol. 3988, Springer, 2006

12 C. Spector, *Provably Recursive Functionals of Analysis: a Consistency Proof of Analysis by an Extension of Principles in Current Intuitionistic Mathematics*, Dekker (ed.), Recursive Function Theory: Proceedings of Symposia in Pure Mathematics, vol. 5. AMS, Providence, 1962

A Non-Standard Semantics for Kahn Networks in Continuous Time*

Romain Beauxis[1] **and Samuel Mimram**[2]

1 **Tulane University, Department of Mathematics**
6823 St Charles Avenue, New Orleans LA 70118, USA
rbeauxis@tulane.edu

2 **CEA, LIST**
Point Courrier 94, 91191 Gif-sur-Yvette, France
samuel.mimram@cea.fr

Abstract

In a seminal article, Kahn has introduced the notion of process network and given a semantics for those using Scott domains whose elements are (possibly infinite) sequences of values. This model has since then become a standard tool for studying distributed asynchronous computations. From the beginning, process networks have been drawn as particular graphs, but this syntax is never formalized. We take the opportunity to clarify it by giving a precise definition of these graphs, that we call *nets*. The resulting category is shown to be a *fixpoint category*, i.e. a cartesian category which is traced wrt the monoidal structure given by the product, and interestingly this structure characterizes the category: we show that it is the free fixpoint category containing a given set of morphisms, thus providing a complete axiomatics that models of process networks should satisfy. We then use these tools to build a model of networks in which data vary over a *continuous* time, in order to elaborate on the idea that process networks should also be able to encompass computational models such as hybrid systems or electric circuits. We relate this model to Kahn's semantics by introducing a third model of networks based on non-standard analysis, whose elements form an *internal complete partial order* for which many properties of standard domains can be reformulated. The use of hyperreals in this model allows it to formally consider the notion of infinitesimal, and thus to make a bridge between discrete and continuous time: time is "discrete", but the duration between two instants is infinitesimal. Finally, we give some examples of uses of the model by describing some networks implementing common constructions in analysis.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases Kahn network, non-standard analysis, fixpoint category, internal cpo

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.35

1 Introduction

Process networks have been introduced by Kahn, together with an associated semantics based on Scott domains, as one of the first model for concurrent and asynchronous computations [19]. These networks are constituted of *processes* (which may be thought as computers on a network or threads on a computer for instance) which perform computations and can exchange information through *channels* acting as unbounded FIFO queues. Finite

* This work was partially supported by the Office of Naval Research and by the PANDA ("Parallel and Distributed Analysis", ANR-09-BLAN-0169) French ANR project.

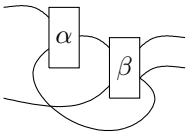
or infinite sequences of values, that are called *streams*, are communicated on the channels, and the semantics of the whole process network is considered to be the streams that can be exchanged, depending on the data provided by the environment. The set of streams can be structured as a complete partial order, and the semantics of networks is modeled by Scott-continuous functions on this domain: the fact that these functions admit a smallest fixpoint turns out to be crucial in order to model "loops" formed by channels in the network. A series of subsequent works have provided a precise understanding of this fixpoint construction [11, 16].

In this model, time is *discrete* in the sense that a countable number of values might be exchanged during an execution: we can consider that each value occurs at a given instant $t \in \mathbb{N}$. In this article, we are interested in understanding how to extend the usual semantics of process networks in order to consider computations in *continuous* time, by replacing $\mathbb{N}$ by $\mathbb{R}^+$ for the domain of time, so as to embrace computational models such as electric circuits or hybrid systems, with which it bears many similarities. However, how would such a semantics relate to the usual discrete semantics of networks? The fundamental intuition in order to relate the two models is the following: if we allow ourselves to consider *infinitesimal* durations $\mathrm{d}t$, then we can think of continuous time as being somehow "discrete", its instants being $0, \mathrm{d}t, 2\,\mathrm{d}t, 3\,\mathrm{d}t, \ldots$ This idea of *infinitesimal time* originates in the works of Bliudze and Krob [7], and was later on developed by Benveniste, Caillaud and Pouzet [6], who formalized it by using tools provided by *non-standard analysis* [27, 12] in order to give a rigorous meaning to infinitesimals. Here, we develop on these ideas by structuring the resulting notion of stream into *internal Scott domains*, which are shown to provide a model of process networks, and explain how the resulting model provides a nice bridge between discrete and continuous time.

For this purpose, we introduce a new model for Kahn process networks. However, what is precisely the syntax for these networks that we want to model? Here, we formalize the definition of the graphs which are often used to informally represent process networks, by a structure that we call *nets*. We show here that the resulting category is a *fixpoint category*, i.e. a cartesian category which is traced [18] wrt the monoidal structure provided by products. Moreover, this structure characterizes the category in the sense that the category of nets is a free category of such kind. This result thus provides a complete description of the axioms that a model of nets should satisfy. We finally use this structure to show that streams in infinitesimal time form such a model. We elaborate here on a series of earlier works. The structure of traced monoidal category for the Kahn model has been introduced by Hildebrandt, Panangaden and Winskel [16] and the construction of nets introduced here is a generalization of the one introduced in [15]. Properties of fixpoint categories and their relationship to fixpoint operators have been studied in details by Hasegawa [14].

We begin by defining nets (Section 2.1), show that they are free fixpoint categories (Section 2.2) and explain that Scott-domain semantics can be given for nets (Section 2.3). We then recall basic constructions and properties of non-standard analysis (Section 3.1), define the notion of internal domain of which infinitesimal-time streams are an instance (Section 3.2) and relate models in infinitesimal and continuous time (Section 3.3). We finally conclude in Section 4.

2 Nets and their semantics



A Kahn process network [19] can be thought as a finite set of boxes, with inputs and outputs, linked through wires, producing outputs depending on their inputs which will be asynchronously transmitted over the wires. Over time, data circulates through the network, producing streams of data. The dataflow semantics of these networks has been well-studied in relation with Scott domains and category theory [25, 16]. However, there is no canonical syntax for them, even though a graphical notation (as pictured on the right) is often used. Since the purpose of this paper is to provide a new semantics for process networks, we take this opportunity to clarify the definition of the syntax, by formalizing the graphical notation and relating it with the categorical structure of the models. The ideas developed here in order to develop an axiomatics for Kahn process networks originate in various previous works in the field. Kahn's original paper [19] mentions results of decidability of the equivalence of graphs representing networks (which are called *schemata*) based on earlier works [9]. Many subsequent articles underline the importance of operations on networks such as sequential composition, parallel composition, tupling (products) and feedback [11, 25], and a traced monoidal category modeling Kahn networks was constructed in [16]. On the categorical side, the "drawings" used here have been formalized as *string diagrams* representing morphisms in monoidal categories [17]. Traced monoidal categories were introduced in [18] and turned out to be a fundamental tool in computer science [1]. Their axiomatics was simplified and studied in the cartesian case [14] and constructions of free traced monoidal categories were provided in [2, 15].

2.1 Nets

A *signature* $\Sigma = (\Sigma, \sigma, \tau)$ consists of a set Σ of *symbols* and two functions $\sigma, \tau : \Sigma \to \mathbb{N}$, which to every symbol α associate its *arity* and *coarity* respectively – we thus sometimes write $\alpha : \sigma(\alpha) \to \tau(\alpha)$: the symbols should be thought as possible building blocks for a process network, with specified number of inputs and outputs. Given such a signature, a net consists of instances of symbols (called operators) whose inputs and outputs are linked together through wires (called ports) defined as follows. Given an integer n, we write $\langle n \rangle$ for the set $\{0, \dots, n-1\}$.

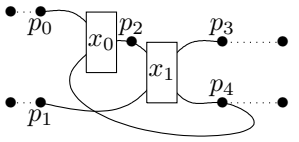
▶ **Definition 1** (Net). A *net* $N = (P, O, \lambda, s, t)$ from m to n, with $m, n \in \mathbb{N}$, consists of
- a finite set P of *ports*,
- a finite set O of *operators*,
- a *labeling function* $\lambda : O \to \Sigma$,
- a *source function* $s : S_N \to P$ and an injective *target function* $t : T_N \to P$, where

$$S_N = \{(x, i) \mid x \in O, i \in \langle \sigma \circ \lambda(x) \rangle\} \uplus \langle n \rangle \qquad T_N = \{(x, i) \mid x \in O, i \in \langle \tau \circ \lambda(x) \rangle\} \uplus \langle m \rangle$$

We sometimes write $N : m \to n$ to indicate that N is a net from m to n.

▶ **Example 2.** Suppose that Σ contains two symbols α and β whose sources (given by σ) are both 2 and targets (given by τ) are respectively 1 and 2. The net drawn in the introduction of this section can be formalized as a net $N : 2 \to 2$ defined by $P = \{p_0, \dots, p_4\}$, $O = \{x_0, x_1\}$, $\lambda(x_0) = \alpha$, $\lambda(x_1) = \beta$, s and t are defined by

$$s(x_0, 0) = p_0 \quad s(x_0, 1) = p_4 \quad s(x_1, 0) = p_2 \quad s(x_1, 1) = p_1 \quad s(0) = p_3 \quad s(1) = p_4$$
$$\text{and} \quad t(x_0, 0) = p_2 \quad t(x_1, 0) = p_3 \quad t(x_1, 1) = p_4 \quad t(0) = p_0 \quad t(1) = p_1$$



Graphically, this can be pictured as on the right. The bullets on the left and on the right indicate the source and target m and n and the dotted lines represent the induced source and target functions $\langle m \rangle \to P$ and $\langle n \rangle \to P$ respectively. Notice that a port can be used as input for multiple wires as it is the case for the port p_4 in the example. However, t being injective, two wires cannot have the same output port.

▶ **Definition 3.** A *morphism* $\varphi : M \to N$ between two nets $M, N : m \to n$ (with the same source and target) consists of a pair of functions $\varphi_P : P_M \to P_N$ and $\varphi_O : O_M \to O_N$ such that for every operator $x \in O_M$, $\lambda_N(\varphi_O(x)) = \lambda_M(x)$, for every source $(x, i) \in S_M$, $\varphi_P(s_M(x, i)) = s_N(\varphi_O(x), i)$, for every $k \in \langle n \rangle$, $\varphi_P(s_N(k)) = s_M(k)$ and similar equations for target functions. Two nets M and N are isomorphic when there exists an invertible morphism between them, which we write $M \approx N$.

▶ **Definition 4.** In order to define a category whose objects are integers and morphisms are nets (considered up to isomorphism), we introduce the following constructions:

- *Identity.* The identity net $N : n \to n$ is the net such that $P = \langle n \rangle$, $O = \emptyset$ and $s, t : \langle n \rangle \to P$ are both the identity function.
- *Composition.* Given two nets $M : n_1 \to n_2$ and $N : n_2 \to n_3$, their composite $N \circ M : n_1 \to n_3$ is the net defined by $P = P_M \uplus P_N / \sim$ where $\sim$ is the smallest equivalence relation such that $s_M(k) \sim t_N(k)$ for every $k \in \langle n_2 \rangle$, $O = O_M \uplus O_N$, $\lambda = \lambda_M \uplus \lambda_N$, s is defined by $s(x, i) = s_M(x, i)$ if $x \in O_M$, $s(x, i) = s_N(x, i)$ if $x \in O_N$ and $s(k) = s_N(k)$ if $k \in \langle n_3 \rangle$, and t is defined similarly.
- *Tensor.* Given two nets $M : m \to m'$ and $N : n \to n'$, their tensor product net $M \otimes N : m + n \to m' + n'$ is the net which is defined by $P = P_M \uplus P_N$, $O = O_M \uplus O_N$, $\lambda = \lambda_M \uplus \lambda_N$, s is defined by $s(x, i) = s_M(x, i)$ if $x \in O_M$ and $s(x, i) = s_N(x, i)$ if $x \in O_N$, $s(k) = s_M(k)$ if $k \in \langle m' \rangle$ and $s(k) = s_N(k - m')$ if $k \in \langle n' \rangle$, and t is defined similarly.
- *Trace.* Given a net $N : n_1 + n \to n_2 + n$, we define the net $\mathrm{Tr}^n_{n_1,n_2}(N) : n_1 \to n_2$ by $P = P_N / \sim$ where $\sim$ is the smallest equivalence relation such that $s_N(n_2 + k) = t_N(n_1 + k)$ for every $k \in \langle n \rangle$, $O = O_N$, $\lambda = \lambda_N$, $s = q \circ s_N$ and $t = q \circ t_N$ where $q : P_N \to P$ is the canonical quotient map.

It can easily be shown that the constructions above are compatible with isomorphisms of nets (e.g. if $M \approx M'$ and $N \approx N'$ then $M \otimes N \approx M' \otimes N'$). It thus makes sense to define the following category:

▶ **Definition 5.** We write $\mathbf{Net}_\Sigma$ for the category $\mathbf{Net}_\Sigma$ whose objects are natural integers, morphisms $N : m \to n$ are isomorphism classes of nets, identities and composition are given by the constructions of Definition 4.

▶ **Lemma 6.** *The category* $\mathbf{Net}_\Sigma$ *is well-defined and is monoidal with the tensor product of Definition 4 and* 0 *as unit.*

▶ Remark. In order to make a more fine-grained study of the categorical structure of nets, we could have avoided quotienting morphisms by isomorphisms of net and defined a bicategory [5] whose 0-cells are integers, 1-cells are nets and 2-cells are morphisms of nets. We did not do this here to simplify the presentation.

▶ Remark. This construction, as well as the following, can be extended without difficulty to define multisorted nets (i.e. where the various inputs of operators have different types), see [15] for a similar construction. A nice and abstract description of this construction can be carried on using polygraphs [8], in a way similar to [24].

Even though the output of an operator can be duplicated or erased, the category $\mathbf{Net}_\Sigma$ fails to have finite products. This is essentially due to the fact that duplication and erasure are not natural, e.g. the following nets (of type $1 \to 2$ and $1 \to 0$ respectively) are different:

We can however recover products by considering the two inequalities above as rewriting rules on nets from left to right as follows.

- *Sharing.* Given a net $N : m \to n$, suppose that there exists two operators $x, y \in O$ with the same label and the same inputs, i.e. $\lambda(x) = \lambda(y)$ and for every $i \in \langle \sigma \circ \lambda(x) \rangle$, $s(x, i) = s(y, i)$. We define a net $N' : m \to n$ by $P = P_N / \sim$ where $\sim$ is the smallest equivalence relation such that $t(x, i) \sim t(y, i)$ for every $i \in \langle \tau \circ \lambda(x) \rangle$, $O = O_N / \sim'$ where $\sim'$ is the smallest equivalence relation identifying x and y, and λ, s and t are obtained by quotienting the corresponding functions of N. The net N' is said to be obtained from N by *sharing*.
- *Erasing.* Given a net $N : m \to n$, suppose that there exists an operator $x \in O$ such that for every $i \in \langle \tau \circ \lambda(x) \rangle$, $s^{-1}(t(x, i)) = \emptyset$. Informally, none of the outputs of the operator x is used as an input for some other operator. We write $N' : m \to n$ for the net obtained from N by removing the operator x as well as all the ports $t(x, i)$ for $i \in \langle \tau \circ \lambda(x) \rangle$. The net N' is said to be obtained from N by *erasing*.

We say that N *se-rewrites* to N' when N' can be obtained from N by sharing or erasing. The *se-equivalence* is the smallest equivalence relation containing the se-rewriting relation.

▶ **Proposition 1.** *The category* $\mathbf{sNet}_\Sigma$ *obtained from* $\mathbf{Net}_\Sigma$ *by quotienting morphisms by se-equivalence has finite products, given on objects by the tensor product of* $\mathbf{Net}_\Sigma$*.*

Proof. The terminal object is 0 and the product of two objects m and n is $m + n$ with the projection of m defined as the net $N : m + n \to m$ such that $P = \langle m + n \rangle$, $O = \emptyset$, $s : \langle m \rangle \to P$ is the canonical injection and $t : \langle m+n \rangle \to P$ is the identity (and the projection on n is defined similarly). All required axioms are easily verified. ◀

It can be shown that the se-rewriting rules form a terminating (they decrease the number of operators) and confluent rewriting system. The normal forms are nets which do not contain two operators with the same label and input ports, and do not contain operators such that none of the outputs are inputs for some other operator. A direct alternative description of nets modulo se-equivalence, called shared nets, can thus be defined as follows.

▶ **Definition 7.** A *shared net* $N = (P, O, s, t)$ from m to n consists of

- a finite set P of *ports*,
- a finite set O of *operators* which are pairs $(\alpha, (s_i)_{i \in \langle \sigma(\alpha) \rangle})$ where $\alpha \in \Sigma$ is a symbol and $(s_i)_{i \in \langle \sigma(\alpha) \rangle}$ is a family of ports called the *sources* of the operators,
- a *source function* $s : \langle n \rangle \to P$,
- an injective *target function* $t : T_N \to P$, where $T_N = \{(x, i) \mid x \in O, i \in \langle \tau \circ \lambda(x) \rangle\} \uplus \langle m \rangle$,

such that for every operator $x \in O$, $s^{-1}(t(T_x)) \neq \emptyset$ where $T_x = \{(x, i) \mid i \in \langle \tau \circ \lambda(x) \rangle\}$.

▶ **Proposition 2.** *A category whose objects are integers and morphisms are shared nets modulo (suitably defined) isomorphism can be defined in a similar way as previously, and this category can be shown to be equivalent to* $\mathbf{sNet}_\Sigma$ *through product-preserving functors.*

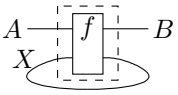
Proof. The canonical forms of nets wrt se-rewriting are in bijection with shared nets. ◀

Next section justifies why the category **sNet** provides a convincing definition of the networks.

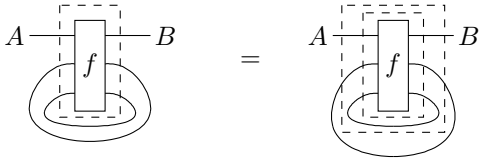
2.2 Nets as free fixpoint categories

We now study the structure of the category $\mathbf{sNet}_\Sigma$ in order to define a proper denotational model this category. Recall that a (strict) *monoidal category* $(\mathcal{C}, \otimes, I)$ consists of a category $\mathcal{C}$ together with a bifunctor $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, called *tensor*, and an object I, called *unit*, such that the tensor is strictly associative and admits units as neutral elements. A (strict) *symmetric monoidal category* consists of a monoidal category $(\mathcal{C}, \otimes, I)$ equipped with a natural family $\gamma_{A,B} : A \otimes B \to B \otimes A$ of isomorphisms satisfying $\gamma_{B,A} \circ \gamma_{A,B} = \mathrm{id}_{A\otimes B}$ as well as other coherence axioms, see [22] for details. Any category $\mathcal{C}$ with finite products admits a structure of symmetric monoidal category with the cartesian product $\times$ as tensor and the terminal object 1 as unit, and this structure can be chosen to be strict in the case of $\mathbf{sNet}_\Sigma$ (thus we only consider strict monoidal categories in the following for simplicity). A natural notion of "feedback" was formalized in monoidal categories by Joyal, Street and Verity [18] as follows:

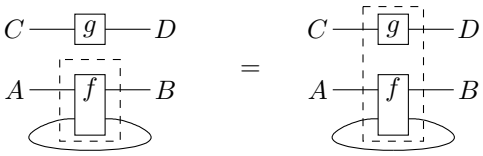
▶ **Definition 8** (Trace)**.** A *trace* on a symmetric monoidal category $\mathcal{C}$ consists of a *natural* family of functions $\mathrm{Tr}^X_{A,B} : \mathcal{C}(A\otimes X, B\otimes X) \to \mathcal{C}(A, B)$. Given a morphism $f : A \otimes X \to B \otimes X$, the morphism $\mathrm{Tr}^X_{A,B}(f) : A \to B$ is often drawn as on the right. A trace should satisfy the following axioms.



1. *Vanishing*: for every $f : A \otimes X \otimes Y \to B \otimes X \otimes Y$, $\mathrm{Tr}^{X\otimes Y}_{A,B}(f) = \mathrm{Tr}^X_{A,B}(\mathrm{Tr}^Y_{A\otimes X, B\otimes X}(f))$



2. *Superposing*:
for every $f : A \otimes X \to B \otimes X$ and $g : C \to D$, $g \otimes \mathrm{Tr}^X_{A,B}(f) = \mathrm{Tr}^X_{C\otimes A, D\otimes B}(g \otimes f)$



3. *Yanking*: for every object X, $\mathrm{Tr}^X_{X,X}(\gamma_{X,X}) = \mathrm{id}_X$

▶ **Proposition 3.** *The construction of Definition 4 induces a trace on* $\mathbf{Net}_\Sigma$ *and on* $\mathbf{sNet}_\Sigma$*.*

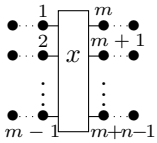
The category $\mathbf{sNet}_\sigma$ is a traced cartesian category that we call a *fixpoint category* in the following. Interestingly, it is actually characterized by this structure in the sense that it is the free fixpoint category containing a Σ-object.

▶ **Definition 9** (Σ-object)**.** Given a signature Σ, a Σ-*object* in a monoidal category $\mathcal{C}$ consists of an object A together with a morphism $f_\alpha : \otimes^{\sigma(\alpha)} A \to \otimes^{\tau(\alpha)} A$ for every symbol $\alpha \in \Sigma$, called the *interpretation* of α, where $\otimes^n A$ denotes the tensor product of n copies of the object A. A Σ-*morphism* between two Σ-objects (A, f_α) and (B, g_α) consists of a morphism $h : A \to B$ such that for every $\alpha \in \Sigma$, $(\otimes^{\tau(\alpha)} h) \circ f_\alpha = g_\alpha \circ (\otimes^{\sigma(\alpha)} h)$.

▶ **Theorem 10.** *The category* $\mathbf{sNet}_\Sigma$ *is the free category containing a* Σ*-object in the sense that for every fixpoint category* $\mathcal{C}$*, the category* $\mathbf{Mod}(\Sigma, \mathcal{C})$ *of* Σ*-objects and* Σ*-morphisms*

is equivalent to the category $\mathbf{Fix}(\mathbf{sNet}_\Sigma, \mathcal{C})$ *whose objects are fixpoint functors (preserving cartesian product and trace) and morphisms are monoidal natural transformations.*

Proof. The category $\mathbf{sNet}_\Sigma$ contains a Σ-object whose underlying object is 1 and the interpretation of a symbol α with $\sigma(\alpha) = m$ and $\tau(\alpha) = n$ is the net $N : m \to n$ such that $P = \langle m + n \rangle$, $O = \{x\}$, $\lambda(x) = \alpha$, $s(x, i) = i$, $s(k) = m + k$, $t(x, i) = m + i$, $t(k) = k$:



A construction of the free traced symmetric monoidal category containing a Σ-object was provided in [2] and reformulated in [15] using a variant of the nets defined here, that we call *traced nets*. It is easy to see that we recover traced nets by restricting $\mathbf{sNet}_\Sigma$ to the nets such that the source function s is a bijective function. We thus have to show that our category of nets is the free category over the category of traced nets. Recall that a cocommutative comonoid (M, δ, ε) in a symmetric monoidal category consists of an object M together with two morphisms $\delta : M \to M \otimes M$ (called *duplication*) and $\varepsilon : M \to I$ (called *erasure*), which are such that $(\delta \otimes \mathrm{id}_I) \circ \delta = (\mathrm{id}_I \otimes \delta) \circ \delta$, $(\varepsilon \otimes \mathrm{id}_I) \circ \delta = \delta = (\mathrm{id}_I \otimes \varepsilon) \circ \delta$ and $\gamma_{M,M} \circ \delta = \delta$. Now, it has been shown [8] that the category whose objects are integers and whose morphisms $f : m \to n$ are functions $f : \langle m \rangle \to \langle n \rangle$ is the free monoidal symmetric monoidal category containing a commutative monoid, and that the free cartesian category over a symmetric monoidal category is obtained by freely adding a natural structure of cocommutative comonoids over each object: precisely, this means that each object M is equipped with a structure $(M, \delta_M, \varepsilon_M)$ of cocommutative comonoid and these are natural in the sense that for every morphism $f : M \to N$, $\delta_N \circ f = (f \otimes f) \circ \delta_M$ and $\varepsilon_N \circ f = \varepsilon_M$. From this it can be deduced that going from nets with bijective s to nets with any function as s, and quotienting by se-equivalence, amounts to construct the free cartesian category over the category of traced nets. Namely, allowing any function equips the object 1 with a structure of comonoid with the duplication δ_1 being the net $N_{\delta_1} : 1 \to 2$ such that $P = \{p\}$, $O = \emptyset$, $s(k) = p$ and $t(k) = p$ and the duplication ε_1 being the net $N_{\varepsilon_1} : 1 \to 0$ such that $P = \{p\}$, $O = \emptyset$ and $t(k) = p$:

N_{δ_1} =  N_{ε_1} =

(and every object can be equipped with a structure of cocommutative comonoid in a similar way). Quotienting by se-equivalence amounts to impose that the structures of cocommutative comonoid on the objects are natural. ◀

2.3 Models of nets

The properties of fixpoint categories have been studied in details by Hasegawa and Hyland [14]. In particular, they have shown that a cartesian category $\mathcal{C}$ is traced if and only if it contains a fixpoint operator satisfying suitable axioms (these are sometimes called *Conway fixpoint operators*). For instance, the category of Scott domains recalled below admits such a fixpoint and is therefore a fixpoint category thus providing a natural semantics for nets.

A *directed complete partial order* (or *dcpo*) is a poset $(D, \leqslant)$ such that every directed subset X has a supremum $\bigvee X$ and a *complete partial order* (or *cpo*) is a dcpo with a least element $\bot$ [3, 4, 10]. A function $f : A \to B$ between two dcpo is *Scott-continuous* when

it preserves supremums. By the Kleene fixpoint theorem, every Scott-continuous function $f : X \to X$ admits a least fixpoint $\mathrm{fix}_X(f)$ defined by $\mathrm{fix}_X(f) = \bigvee_{n\in\mathbb{N}} f^n(\bot_X)$, where f^n denotes the composite of n instances of f. Suppose given a function $f : A \times X \to B \times X$. We write $\pi_B : B \times X \to B$ and $\pi_X : B \times X \to X$ for the canonical projections. Given $a \in A$, we write $f_a = x \mapsto f(a, x)$ for the partial application of f to a. A trace can defined on f by

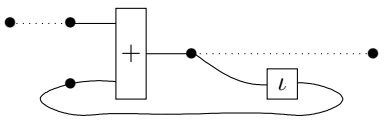
$$\mathrm{Tr}^X_{A,B}(f) \quad = \quad a \quad \mapsto \quad \pi_B(\mathrm{fix}_{B\times X}(f_a \circ \pi_X)) \tag{1}$$

and this function can be shown to be Scott-continuous.

▶ **Proposition 4.** *The category* **Cpo** *of cpo and Scott-continuous functions is a fixpoint category with* (1) *as trace.*

By Theorem 10, any Σ-object in **Cpo** canonically induces a functor $F : \mathbf{Net}_\Sigma \to \mathbf{Cpo}$ which associates to every net its *domain semantics*: once we have interpreted each symbol as a Scott-continuous function (by fixing a Σ-object), the interpretation of each network is also fixed. In particular, when the Σ-object is the domain R^∞ of R-valued streams (for some set R), we recover the usual Kahn semantics [19] of nets: R^∞ is the domain whose elements are finite or infinite sequences (called *streams*) of elements of R, ordered by inclusion. The intuition here is that time is discrete (because we only consider the times where some information is passed on a wire as *instants*) and the elements of the domain are the sequences of values passed on wires at the various instants.

▶ **Example 11.** Consider a signature Σ containing two symbols $+ : 2 \to 1$ and $\iota : 1 \to 1$. We consider the Σ-object $\mathbb{R}^\infty$ in **Cpo** such that the interpretation of $+$ is the Scott-continuous function $\mathbb{R}^\infty \times \mathbb{R}^\infty \to \mathbb{R}^\infty$ such that the image of two streams of same length is their pointwise addition and the interpretation of ι is the function $\mathbb{R}^\infty \to \mathbb{R}^\infty$ which prepends a 0 to streams. The interpretation of the net on the right is the function which returns the stream whose n-th value is the sum of the $n + 1$ first values of the stream.



An element of the Kahn domain can be considered as a partial function $s : \mathbb{N} \to R$ whose domain of definition is an initial segment of $\mathbb{N}$ (the integers in $\mathbb{N}$ represent the instants of the time). Our goal here is to consider a semantics where *time is continuous*, i.e. streams are generalized to partial functions $s : \mathbb{R}^+ \to R$ defined on an initial segment of $\mathbb{R}^+$ and to relate it to the Kahn semantics. In order to build bridge between the two models, the intuition here is that continuous time can be considered as "discrete" if we allow ourselves to consider *infinitesimals*: the time in $\mathbb{R}^+$ can namely be thought as a sequence of instants $0, \mathrm{d}t, 2\,\mathrm{d}t, 3\,\mathrm{d}t, \ldots$ where $\mathrm{d}t$ is an infinitesimal, thus enabling us to extend techniques developed for Kahn networks to continuous time semantics. Moreover, many operations of common use in analysis can be simply formulated by an appropriate net with the continuous time semantics. For instance, the derivative of a function whose definition can be formulated as $f'(t) = (f(t) - f(t - \mathrm{d}t))/\,\mathrm{d}t$ can be implemented by a net of the form (4) which directly translates to nets the above formula. The rest of the paper is devoted to explaining and formalizing these ideas by using of non-standard analysis which allows us to rigorously make sense of the notion of infinitesimal.

3 A non-standard semantics for Kahn networks in continuous time

3.1 Hyperreals

We give here a brief introduction to non-standard analysis and refer the reader to textbooks [12, 27, 26] for details. The motivation underlying the construction of hyperreals is to extend the field $\mathbb{R}$ of real numbers into a field ${}^*\mathbb{R}$ in which one can give a meaning to the notions of infinitesimal and infinite numbers. Basically, hyperreal numbers are defined as countable sequences $(x_i)_{i\in\mathbb{N}}$ of real numbers, the sequences converging towards 0 representing infinitesimals. Any real x can be seen as the hyperreal which is the constant sequence whose elements are equal to x, moreover the usual operations are extended pointwise to hyperreals, e.g. the multiplication is defined by $(x_i) \times (y_i) = (x_i \times y_i)$. In order for suitable axioms to be satisfied (for instance every non-null hyperreal should have an inverse) one has to consider equivalence classes of such sequences; in particular, any two sequences which only differ on a finite number of values should be equivalent. The starting point of non-standard analysis is the fact that a suitable equivalence relation can be defined from an ultrafilter:

▶ **Definition 12** (Ultrafilter)**.** A *filter* on a set I is a collection $\mathcal{F}$ of sets which is closed under intersection and under supersets (i.e. if $U \subseteq V \subseteq I$ and $U \in \mathcal{F}$ then $V \in \mathcal{F}$). A filter is *proper* when $\emptyset \notin \mathcal{F}$. An *ultrafilter* on I is a proper filter such that for every $U \subseteq I$, either $U \in \mathcal{F}$ or $I \setminus U \in \mathcal{F}$. An ultrafilter $\mathcal{F}$ is *principal* when there exists $i \in I$ such that $\mathcal{F} = \{U \subseteq I \mid i \in U\}$, and *non-principal* otherwise.

Assuming Zorn's lemma (which is equivalent to the axiom of choice), it can be shown that

▶ **Proposition 5.** *Any infinite set I has a non-principal ultrafilter on it.*

We now fix such an ultrafilter $\mathcal{F}$ on $\mathbb{N}$ whose elements are called *large sets*. The fact that $\mathcal{F}$ is non-principal implies that it does not contain any finite subset of $\mathbb{N}$: the construction of the ultrafilter can thus be thought as a way of constructing a set, starting from all cofinite sets, and coherently adding either I or its complement for every set $I \subseteq \mathbb{N}$ which is neither finite nor cofinite. We define an equivalence relation $\equiv$ on countable sequences of reals by $(x_i) \equiv (y_i)$ when $\{i \in \mathbb{N} \mid x_i = y_i\} \in \mathcal{F}$ and denote by $\langle x_i \rangle$ the equivalence class of (x_i).

▶ **Definition 13** (Hyperreals)**.** The set ${}^*\mathbb{R}$ of *hyperreals* is the set of equivalence classes (wrt $\equiv$) of countable sequences of reals.

The set ${}^*\mathbb{N}$ of *hyperintegers* is defined similarly and there is a canonical inclusion ${}^*\mathbb{N} \subseteq {}^*\mathbb{R}$.

Any countable sequence (x_i) of reals induces an hyperreal $\langle x_i \rangle$, and in particular a real r can be seen as an hyperreal ${}^*r = \langle r \rangle$ by considering the equivalence class of the constant sequence whose elements are equal to r (we sometimes leave this conversion implicit). Similarly, a countable sequence (X_i) of subsets of $\mathbb{R}$ induces a set $\langle X_i \rangle$ of hyperreals defined as the set of $\langle x_i \rangle \in {}^*\mathbb{R}$ such that $\{i \in \mathbb{N} \mid x_i \in X_i\} \in \mathcal{F}$. A subset of ${}^*\mathbb{R}$ is an *internal set* if it can be obtained this way, in particular any set $X \subseteq \mathbb{R}$ induces an internal set ${}^*X = \langle X \rangle$, associated to the constant sequence (for instance $\langle \mathbb{R} \rangle = {}^*\mathbb{R}$). Similarly, a countable sequence of functions $(f_i : A_i \to B_i)$, where the A_i and B_i are subsets of $\mathbb{R}$, extends to a function $\langle f_i \rangle : \langle A_i \rangle \to \langle B_i \rangle$, defined on $\langle x_i \rangle \in \langle A_i \rangle$ by $\langle f_i \rangle(\langle x_i \rangle) = \langle \overline{f_i}(x_i) \rangle$ where $\overline{f_i}(x_i) = f_i(x_i)$ if $x_i \in A_i$ and $\overline{f_i}(x_i) = 0$ otherwise. Such a function is called an *internal function*. Any real-valued function $f : A \to B$ may be seen as an internal function ${}^*f = \langle f \rangle : \langle A \rangle \to \langle B \rangle$. The notion of *internal relation* is defined similarly.

▶ Remark. As we explain in Section 3.1.1, it is important to keep in mind that not every set $X \subseteq {}^*\mathbb{R}$ (or function, or relation) is internal.

Notice that in the above definition of an internal function, we have used 0 as "default value" for the functions f_i on the elements $x_i \notin A_i$. This could be avoided by choosing a suitable representative in the equivalence class $\langle x_i \rangle$:

▶ **Lemma 14.** *Given an element x of an internal set $\langle A_i \rangle$, there exists a sequence (y_i), such that $y_i \in A_i$ for every index i, satisfying $\langle y_i \rangle = x$.*

In the way described above, all the usual operations on reals extend to hyperreals (and similarly for hyperintegers). For instance, the absolute value of an hyperreal $\mathtt{x} = \langle x_i \rangle$ is defined by $|\mathtt{x}| = \langle |x_i| \rangle$. An hyperreal $\mathtt{x}$ of ${}^*\mathbb{R}$ is *infinitesimal* whenever $|\mathtt{x}| < r$ for every real $r > 0$, and *unlimited* if $r < |\mathtt{x}|$ for every real $r \in \mathbb{R}$. Given a hyperreal $\mathtt{x}$ which is not unlimited, there exists a unique real y such that $\mathtt{x} - y$ is infinitesimal: this real is called the *standard part* of $\mathtt{x}$ and denoted by $\mathrm{st}(\mathtt{x})$. We define an equivalence relation $\approx$ on hyperreals by $\mathtt{x} \approx \mathtt{y}$ whenever $\mathrm{st}(\mathtt{x} - \mathtt{y}) = 0$.

▶ Remark. The existence of a standard part might be surprising at first: for instance, given the sequence x_i such that $x_i = 0$ if i is even and $x_i = 1$ otherwise, what should be the standard part of $\langle x_i \rangle$? The result is given by the chosen ultrafilter $\mathcal{F}$: if the set I of even integers is in $\mathcal{F}$ then $\mathrm{st}(\langle x_i \rangle) = 0$, otherwise the set $\mathbb{N} \setminus I$ of odd integers is in $\mathcal{F}$ and $\mathrm{st}(\langle x_i \rangle) = 1$.

▶ Remark. The method used to construct ${}^*\mathbb{R}$ is an instance of a very general construction of ultraproducts in model theory, which can be used to define a non-standard model ${}^*\mathbb{X}$ from any model $\mathbb{X}$ [21, 27, 12]. In particular, given sets $\mathbb{X}$ and $\mathbb{Y}$, this construction applied to the set $\mathbb{Y}^{\mathbb{X}}$ of functions from $\mathbb{X}$ to $\mathbb{Y}$ constructs the set ${}^*(\mathbb{Y}^{\mathbb{X}})$ of internal functions from ${}^*\mathbb{X}$ to ${}^*\mathbb{Y}$.

3.1.1 The transfer principle

A fundamental tool in non-standard analysis is the *transfer principle*, which follows from Łoś theorem [21]. Informally, this principle can be formulated as follows

▶ **Proposition 6** (Transfer principle)**.** *A first-order formula φ is satisfied in $\mathbb{R}$ if and only if it is satisfied in ${}^*\mathbb{R}$, if we assume that all the sets, functions and relations involved in the formula are internal.*

A similar theorem can be formulated for ${}^*\mathbb{N}$. Many constructions of standard analysis can thus be transferred to non-standard analysis. For instance, the sets ${}^*\mathbb{N}$ and ${}^*\mathbb{R}$ are, respectively, a ring and a field and both are totally ordered.

▶ Remark. The assumption that we consider only internal objects is very important. For instance the formula $((\forall x \in A.\ x \in \mathbb{N}) \wedge (\exists x \in \mathbb{N}.\ x \in A)) \Rightarrow (\exists x \in A. \forall y \in A.\ x \leqslant y)$ is true in $\mathbb{N}$: every non-empty subset A of $\mathbb{N}$ admits a smallest element. From this, we can deduce by transfer that every non-empty *internal* subset of ${}^*\mathbb{N}$ admits a smallest element. However, this property does not hold for every subset of ${}^*\mathbb{N}$: for instance, ${}^*\mathbb{N} \setminus \mathbb{N}$ does not have a smallest element since it can be shown to be closed under predecessor, and is thus not internal. Likewise a subset of ${}^*\mathbb{N}$ (resp. ${}^*\mathbb{R}$) is internal if and only if it is finite.

3.1.2 Non-standard analysis

One of the most interesting property of hyperreals is that it allows one to rigorously consider infinitesimals and thus formalize in an elegant way many of the tools in common use in standard analysis. We give below some of these reformulations which will be of use afterward.

▶ **Proposition 7** (Continuity). *A function $f : \mathbb{R} \to \mathbb{R}$ is* continuous *at x when for every $\mathtt{y} \in {}^*\mathbb{R}$ such that $\mathtt{y} \approx x$, we have ${}^*f(\mathtt{y}) \approx f(x)$. Otherwise said, f is continuous at x when for every infinitesimal $\delta \approx 0$, there exists an infinitesimal $\varepsilon \approx 0$ such that ${}^*f(x+\delta) = f(x)+\varepsilon$.*

▶ **Proposition 8** (Differentiation). *A function $f : \mathbb{R} \to \mathbb{R}$ admits $y \in \mathbb{R}$ as* derivative *at $x \in \mathbb{R}$ when for every non-null infinitesimal $\delta \approx 0$, we have $({}^*f(x+\delta) - f(x))/\delta \approx y$. Furthermore, if f is continuously differentiable on $\mathbb{R}$, then for any two non-null distinct infinitesimals δ and ε, and for any $x \in \mathbb{R}$, we have*

$$f'(x) \quad = \quad \mathrm{st}\left(\frac{{}^*f(x+\delta) - {}^*f(x+\varepsilon)}{\delta - \varepsilon}\right)$$

Given a continuous function $f : \mathbb{R} \to \mathbb{R}$, its integral on an interval $[a, b]$ is defined by

$$\int_a^b f(x)\,\mathrm{d}x \quad = \quad \lim_{n\to\infty}\left(\sum_{k=0}^{n-1} f\left(a + \frac{k}{n}(b-a)\right)\frac{1}{n}\right)$$

Notice that each sum makes sense because it is finite since it is indexed over the finite set $\{k \in \mathbb{N} \mid 0 \leqslant k < n\}$. This notion of finite set can be generalized to internal sets as follows: an internal set $A = \langle A_i \rangle$ is *hyperfinite* if almost all the A_i are finite, i.e. $\{i \in \mathbb{N} \mid A_i \text{ is finite}\} \in \mathcal{F}$. By an argument similar to Lemma 14, we can always suppose that all the A_i are finite by choosing a suitable sequence of finite sets B_i such that $\langle B_i \rangle = \langle A_i \rangle$. Given such an internal set and an internal function $\langle f_i \rangle$, we define $\sum_{\langle x_i \rangle \in \langle A_i \rangle} \langle f_i \rangle(\langle x_i \rangle) = \langle \sum_{x_i \in A_i} f_i(x_i) \rangle$.

▶ **Proposition 9** (Integration). *Given a function $f : \mathbb{R} \to \mathbb{R}$ which is continuous on an interval $[a, b]$, excepting possibly a finite number of points of discontinuity, we have*

$$\int_a^b f(x)\,\mathrm{d}x \quad = \quad \mathrm{st}\left(\sum_{x \in N} {}^*f(a + x(b-a))\delta\right) \tag{2}$$

where $\delta = 1/\mathtt{n}$ for some unlimited $\mathtt{n} = \langle n_i \rangle \in {}^*\mathbb{N}$ and N is the hyperfinite set $N = \langle N_i \rangle \subseteq {}^*\mathbb{R}$ with $N_i = \{k_i/n_i \in \mathbb{R} \mid k_i \in \mathbb{N}, 0 \leqslant k_i < n_i\}$. In particular, the result does not depend on the choice of the unlimited hyperinteger $\mathtt{n} \in {}^*\mathbb{N}$.

The notion defined above corresponds to the Riemann integral. More refined notions (such as the Lebesgue integral) can also be adapted to the non-standard setting.

3.2 Internal domains

In this section, we introduce the notion of internal domain, which we use to define a non-standard denotational semantics for process networks. Given a totally ordered set T and a set R, we write $R^{\leqslant T}$ for the set of partial functions $s : T \to R$ defined on an initial segment of T, called the *domain of definition* of s. The elements of $R^{\leqslant T}$ are called *streams*: the set T can be thought as *time* and the elements of R as the possible *values* of a stream over time. Every such set can be equipped with a partial order $\sqsubseteq$ such that, given $r, s \in R^{\leqslant T}$, we have $f \sqsubseteq g$ whenever the definition domain of r is included in the definition domain of s and r and s coincide on the domain of definition of r.

▶ **Proposition 10.** *The poset* $(R^{\leqslant T}, \sqsubseteq)$ *is a cpo with smallest element* $\bot$ *being the function nowhere defined.*

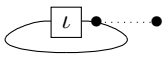
▶ **Example 15.** The Kahn domain described in Section 2.3 is $R^{\leqslant \mathbb{N}}$.

Every function $f : R \to R$ lifts to a function $\tilde{f} : R^{\leqslant T} \to R^{\leqslant T}$ such that, given $s \in R^{\leqslant T}$, the domain of definition of $\tilde{f}(s)$ is the same as the domain of definition of s and the image of s is defined by $\tilde{f}(s)(t) = f \circ s(t)$. The function $\tilde{f}$ is called the *lifting* of f to $R^{\leqslant T}$. It is easy to show that every such lifting is Scott-continuous.

In the following, we will be interested in modeling nets operating in a time which varies continuously. We thus introduce the following domain in order to model the data flowing through the wires:

▶ **Definition 16** (Continuous-time domain)**.** The *continuous-time domain* is the complete partial order $CT = \mathbb{R}^{\leqslant \mathbb{R}^+}$. The *continuous-time domain of continuous functions* CCT is the subdomain of CT whose elements are continuous partial functions $\mathbb{R}^+ \to \mathbb{R}$.

As explained in the introduction, the purpose of this paper is to explain how to implement Scott-continuous functions over this domain using Kahn networks by formalizing the following intuition using non-standard semantics: continuous time can be considered as "discrete" where the duration between two instants is infinitesimal. A natural candidate for this would be the domain ${}^*\mathbb{R}^{\leqslant {}^*\mathbb{N}}$. Namely, in the view of Proposition 9, we would like to relate a stream $s \in \mathbb{R}^{\leqslant \mathbb{R}^+}$ with the stream $\overline{s} \in {}^*\mathbb{R}^{\leqslant {}^*\mathbb{N}}$ defined on $\mathtt{t} \in {}^*\mathbb{N}$ by $\overline{s}(\mathtt{t}) = {}^*s(\mathtt{t}\delta)$, from some infinitesimal $\delta \in {}^*\mathbb{R}$. It turns out that the fixpoints computed in ${}^*\mathbb{R}^{\leqslant {}^*\mathbb{N}}$ are not satisfactory. For instance, consider the net on the right such that the interpretation of the operator ι is the function $\iota : {}^*\mathbb{R}^{\leqslant {}^*\mathbb{N}} \to {}^*\mathbb{R}^{\leqslant {}^*\mathbb{N}}$ such that the image of a stream r is the stream s defined by $s(0) = 0$ and for any non-null hyperinteger $\mathtt{t}$, $s(\mathtt{t}) = r(\mathtt{t} - 1)$. We expect the interpretation of this net to be the constant function equal to 0. However, this is not the case: the semantics s of this net is given by the fixpoint $s = \mathrm{fix}(\iota) = \bigvee_{k \in \mathbb{N}} \iota^k(\bot)$ of ι. Given $k \in \mathbb{N}$, the domain of definition of the stream $\iota^k(\bot)$ is the set $\{\mathtt{p} \in {}^*\mathbb{N} \mid 0 \leqslant \mathtt{p} < k\}$. Therefore, given an unlimited $\mathtt{n} \in {}^*\mathbb{N}$, $\iota^k(\bot)(\mathtt{n})$ is undefined for every $k \in \mathbb{N}$ and thus $\mathrm{fix}(\iota)(\mathtt{n})$ is undefined. Intuitively, the induction on $k \in \mathbb{N}$ defining the smallest fixpoint is not powerful enough to reach all elements of ${}^*\mathbb{N}$. The cpo ${}^*\mathbb{R}^{\leqslant {}^*\mathbb{N}}$ is thus not the appropriate domain, however we explain below that internal domains are a more suitable notion, because they support an induction principle on ${}^*\mathbb{N}$.



(3)

▶ **Definition 17** (Internal cpo)**.** An *internal cpo* $(D, \leqslant)$ in a non-standard model consists of an internal set $D = \langle D_i \rangle$ and an internal relation $\leqslant = \langle \leqslant_i \rangle$ such that for every integer i, $(D_i, \leqslant_i)$ is a cpo. Similarly, an *internal Scott-continuous function* $f : D \to E$ between two internal cpo $D = \langle D_i \rangle$ and $E = \langle E_i \rangle$ consists of an internal function $\langle f_i : D_i \to E_i \rangle$ such that all the f_i are continuous. We write **ICpo** for the category of internal cpo and internal Scott-continuous functions.

▶ Remark. Notice that such an internal cpo $(D, \leqslant)$ is not necessarily a cpo: only internal directed subsets are required to have a supremum. For instance suppose fixed an unlimited hyperinteger $\mathtt{n} \in {}^*\mathbb{N}$. The set $D = \{\mathtt{k} \in {}^*\mathbb{N} \mid \mathtt{k} \leqslant \mathtt{n}\}$ equipped with the usual total order is an internal cpo, but not a cpo because the (non-internal) subset $\mathbb{N} \subseteq D$ is directed and does not have a supremum.

▶ **Proposition 11.** *Any internal Scott-continuous function* $f : D \to D$*, where* D *is an internal cpo, admits a least fixpoint* $\mathrm{fix}(f)$ *which satisfies* $\mathrm{fix}(f) = \bigvee\{f^{\mathtt{n}}(\bot) \mid \mathtt{n} \in {}^*\mathbb{N}\}$*. Here, if* $f = \langle f_i \rangle$ *and* $\mathtt{n} = \langle n_i \rangle$*,* $f^{\mathtt{n}}$ *is defined as* $\langle f_i^{n_i} \rangle$*.*

The axioms of fixpoint categories can be formulated in first-order logic. Using the transfer principle (Proposition 6), it can be shown that the fact that **Cpo** is a fixpoint category implies that

▶ **Proposition 12.** *The category* **ICpo** *is a fixpoint category.*

In the category **ICpo**, we will be particularly interested in the following domain:

▶ **Definition 18** (Infinitesimal-time domain)**.** The *infinitesimal-time domain* is the internal complete partial order $IT = {}^*(\mathbb{R}^{\leqslant \mathbb{N}})$.

As explained in the remark in the end of Section 3.1, the elements of $IT = {}^*(\mathbb{R}^{\leqslant \mathbb{N}})$ are the internal partial functions from ${}^*\mathbb{N}$ to ${}^*\mathbb{R}$. The order $\sqsubseteq$ on this domain is such that for any $r, s \in IT$, we have $r \sqsubseteq s$ whenever the domain of definition of r is included in the domain of definition of s and r and s coincide on the domain of definition of r.

3.3 Comparing continuous time and infinitesimal time

In this section, we explain how the semantics of nets in IT can "simulate" operations in CT. We now suppose fixed an infinitesimal δ called *sampling period*. We define a function $S : CT \to IT$, called *sampling*, which to every stream $s \in CT$ associates the stream $S(s) = \mathtt{k} \mapsto {}^*s(\mathtt{k}\delta)$, and a function $T : IT \to CT$, called *standardization*, which to every stream s associates $T(s) = x \mapsto \mathrm{st}\,(s(\lfloor {}^*x/\delta \rfloor))$, where $\lfloor - \rfloor : {}^*\mathbb{R} \to {}^*\mathbb{N}$ denotes the floor function, and is defined on the biggest initial segment of $\mathbb{R}^+$ for which this definition makes sense. These functions enable us to show that CCT (the domain of *continuous* streams) is a retract of IT. We discuss afterward the possible extensions of this result to elements of CT.

▶ **Proposition 13.** *The restriction of the composite $T \circ S$ to CCT is the identity.*

Proof. Suppose given a stream $s \in CCT$. For any $x \in \mathbb{R}^+$, the fact that s is continuous at x implies, by Proposition 7, that for every $\mathtt{k} \in {}^*\mathbb{N}$ such that $\mathtt{k}\delta \approx x$, we have $S(s)(\mathtt{k}\delta) \approx s(x)$. From this we deduce that $T(S(s))(x) = s(x)$. ◀

▶ Remark. The function $T \circ S$ is generally *not* the identity on CT. For instance, suppose that $\delta = 1/\mathtt{n}$, where $\mathtt{n} \in {}^*\mathbb{N}$ is unlimited, and consider the stream $s \in CT$ whose value is 0 everywhere except at $\sqrt{2}$ where $s(\sqrt{2}) = 1$. Using the transfer principle, it is easy to show that for every $\mathtt{k} \in {}^*\mathbb{N}$, we have $\mathtt{k}/\mathtt{n} \neq \sqrt{2}$. From this we can deduce that $T \circ S(s)$ is the constant stream equal to 0.

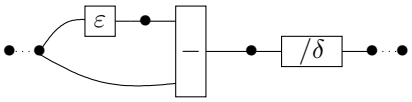
In order to make a more convincing case of the interest of the domain IT as a model of nets and study further its relationship with CT, we give below some examples of nets interpreted in IT which implement common constructions in analysis, and relate them to the corresponding constructions in CT through S and T. For concision, we do not detail the easy verification that the interpretations of operators are internal Scott-continuous functions.

As a first simple example, consider the net (3). From the characterization of the fixpoint of internal Scott-continuous functions given by Proposition 11, it is easy to check that its semantics s in the domain IT is the constant function (defined everywhere) as expected: if we write $c_0 : \mathbb{R}^+ \to \mathbb{R}$ for the constant function equal to 0, we have $s = S(c_0)$ and $c_0 = T(s)$.

3.3.1 Differentiation

The *differentiation* is the following net where "ε" is interpreted as the function which drops the first element of a stream (i.e. $\varepsilon(s)(\mathtt{k}) = s(\mathtt{k}+1)$), "$-$" is interpreted as the pointwise

difference (i.e. $(-(s,t))(\mathtt{k}) = s(\mathtt{k}) - t(\mathtt{k})$), and "$/\delta$" is interpreted as the pointwise division by δ (i.e. $(s/\delta)(\mathtt{k}) = s(\mathtt{k})/\delta$).



(4)

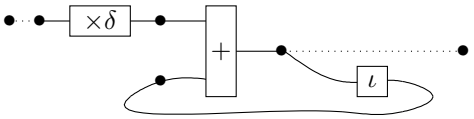
We write $\varphi : IT \to IT$ for the semantics of the net. By Proposition 8, we have immediately:

▶ **Proposition 14.** *For any continuously differentiable function $s : \mathbb{R}^+ \to \mathbb{R}$ in CCT, $T(\varphi(S(s))) = s'$.*

Proof. Given $\mathtt{k} \in {}^*\mathbb{N}$ such that $\mathrm{st}(\mathtt{k}\delta) \in \mathbb{R}^+$, $\varphi(S(s))(\mathtt{k}) = \frac{S(s)(\mathtt{k}+1) - S(s)(\mathtt{k})}{\delta} \approx s'(\mathrm{st}(\mathtt{k}\delta))$. The second step is proved by Proposition 8. Therefore, $T(\varphi(S(s))) = s'$. ◀

3.3.2 Integration

The *integration* is the following net where "$\times\delta$" is the pointwise multiplication of a stream by δ, "$+$" is the pointwise addition of streams, and ι prepends 0 to a stream (see Section 3.2).



(5)

We write $\varphi : IT \to IT$ for the semantics of the net. By Proposition 9, we have immediately:

▶ **Proposition 15.** *For any function $s : \mathbb{R}^+ \to \mathbb{R}$ in CCT, $T(\varphi(S(s))) = x \mapsto \int_0^x s(t)\,\mathrm{d}t$.*

Proof. The semantics φ of the net is computed by a fixpoint as explained in Section 2, defined by $\varphi(S(s))(0) = \delta S(s)(0)$ and $\varphi(S(s))(\mathtt{n}+1) = \varphi(S(s))(\mathtt{n}) + \delta S(s)(\mathtt{n}+1)$. Therefore we have $\varphi(S(s))(\mathtt{n}) = \delta \sum_{\mathtt{k}=0}^{\mathtt{n}} S(s)(\mathtt{k})$. Finally, by Proposition 9, it can be shown that if $\mathrm{st}(\mathtt{x}\delta) = x \in \mathbb{R}^+$, then $\delta \sum_{\mathtt{k}=0}^{\mathtt{n}} S(s)(\mathtt{k}) \approx \int_0^x s(t)\,\mathrm{d}t$ and thus $T(\varphi(S(s)))(x) = \int_0^x s(t)\,\mathrm{d}t$. ◀

This construction can be generalized in order to describe solvers for ordinary differential equations [7, 6]. It should be noticed that the above propositions show that the choice of the infinitesimal sampling period δ is essentially irrelevant.

Most of the preceding results can be adapted to the case where the streams considered are only piecewise continuous, with a finite number of discontinuities. In particular, for any such stream s we have $T \circ S(s) \hat{=} s$ where $\hat{=}$ denotes the equality almost everywhere (this weakening of equality is necessary because of phenomena such as the one described in the remark following Proposition 13). However, the formalization of this is obscured by the fact that piecewise continuous functions, with a finite number of discontinuities, do not form a cpo because the supremum in CT of a directed set of such functions might have an infinite number of points of discontinuity: this is sometimes referred to as the *Zeno phenomenon* in the study of hybrid systems.

4 Conclusion and future works

We have defined nets which provide a formal syntax for process networks, studied the categorical structure of their models, and constructed the infinitesimal-time model as an internal cpo. The fascinating links between denotational semantics of concurrent systems and non-standard analysis have started to be explored only recently and many structures are still yet to be clarified. As explained above, the study of the infinitesimal-time domain has to be refined in order to cope with streams which are not necessarily continuous, and thus to

properly model full-fledged hybrid systems. In particular, Proposition 13 fails to be true if we replace CCT by CT: we plan to investigate generalizations of this property where S and T form an adjunction. We are also investigating possible adaptations of nets in the case where we consider a synchronous semantics (in which there is a notion of simultaneity of events). In this setting, the usual delay operator can elegantly be modeled in feedback categories [20] (which are traced monoidal categories with the yanking axiom removed) and we plan to study nets for those categories. Finally, we envisage many connections with other areas of denotational semantics. For instance, the trace semantics of Kahn networks is closely related to game semantics [23] and it is thus natural to wonder if non-standard analysis can provide insights about a possible definition of a "continuous game semantics" in the spirit of the geometric models for concurrent programs [13].

Acknowledgments: the authors would like to warmly thank Simon Bliudze, Paul-André Melliès, Michael Mislove and Marc Pouzet for all the discussions they had, directly or indirectly related to the subject of this article, as well as the anonymous referees for helpful remarks.

References

1. S. Abramsky. Retracing some paths in process algebra. *CONCUR'96*, pages 1–17, 1996.
2. S. Abramsky. Abstract scalars, loops, and free traced and strongly compact closed categories. *Algebra and Coalgebra in Computer Science*, pages 1–29, 2005.
3. S. Abramsky and A. Jung. *Domain theory*. Oxford University Press, Oxford, UK, 1994.
4. R.M. Amadio and P.L. Curien. *Domains and lambda-calculi*. C.U.P., 1998.
5. J. Bénabou. Introduction to bicategories. In *Reports of the Midwest Category Seminar*, pages 1–77. Springer, 1967.
6. A. Benveniste, B. Caillaud, and M. Pouzet. The Fundamentals of Hybrid Systems Modelers. In *49th IEEE International Conference on Decision and Control (CDC)*, 2010.
7. S. Bliudze and D. Krob. Modelling of complex systems: Systems as dataflow machines. *Fundamenta Informaticae*, 91(2):251–274, 2009.
8. A. Burroni. Higher-dimensional word problems with applications to equational logic. *Theoretical Computer Science*, 115(1):43–62, 1993.
9. B. Courcelle, G. Kahn, and J. Vuillemin. Algorithmes d'equivalence et de reduction a des expressions minimales dans une classe d'equations recursives simples. *Automata, Languages and Programming*, pages 200–213, 1974.
10. B.A. Davey and H.A. Priestley. *Introduction to lattices and order*. C.U.P., 2002.
11. A. Faustini. An operational semantics for pure dataflow. *Automata, Languages and Programming*, pages 212–224, 1982.
12. R. Goldblatt. *Lectures on the Hyperreals*. Springer New York, 1998.
13. E. Goubault. Geometry and concurrency: a user's guide. *Mathematical Structures in Computer Science*, 10(04):411–425, 2000.
14. M. Hasegawa. Recursion from cyclic sharing: traced monoidal categories and models of cyclic lambda calculi. *Typed Lambda Calculi and Applications*, pages 196–213, 1997.
15. M. Hasegawa, M. Hofmann, and G. Plotkin. Finite dimensional vector spaces are complete for traced symmetric monoidal categories. In *Pill. of c. sci.*, pages 367–385. Springer, 2008.
16. T.T. Hildebrandt, P. Panangaden, and G. Winskel. A relational model of non-deterministic dataflow. *Mathematical Structures in Computer Science*, 14(05):613–649, 2004.
17. A. Joyal and R. Street. The geometry of tensor calculus. I. *Adv. in Math.*, 88(1):55–112, 1991.
18. A. Joyal, R. Street, and D. Verity. Traced monoidal categories. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 119, pages 447–468. C.U.P., 1996.

19 G. Kahn. The semantics of a simple language for parallel programming. *Information processing*, 74:471–475, 1974.

20 P. Katis, N. Sabadini, and RFC Walters. On the algebra of feedback and systems with boundary. *Rendiconti del Seminario Matematico di Palermo*, 1999.

21 J. Łoś. The algebraic treatment of the methodology of elementary deductive systems. *Studia Logica*, 2(1):151–211, 1955.

22 S. MacLane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer Verlag, 1971.

23 P.A. Melliès and S. Mimram. Asynchronous games: innocence without alternation. *CONCUR'07*, pages 395–411, 2007.

24 S. Mimram. Computing Critical Pairs in 2-Dimensional Rewriting Systems. In *Proceedings of the 21st RTA*, volume 6 of *LIPIcs*, pages 227–242, 2010.

25 P. Panangaden, V. Shanbhogue, and E. Stark. Stability and sequentiality in dataflow networks. *Automata, Languages and Programming*, pages 308–321, 1990.

26 A. Pétry. Balade en Analyse non standard sur les traces de A. Robinson. *Bulletin of the Belgian Mathematical Society Simon Stevin*, 3(3):1–28, 1996.

27 A. Robinson. *Non-standard analysis*. Princeton University Press, 1996.

Filter models: non-idempotent intersection types, orthogonality and polymorphism

Alexis Bernadet[1,2] **and Stéphane Lengrand**[1,3]

1 **École Polytechnique, France**
2 **École Normale Supérieur de Cachan, France**
3 **CNRS, France**

Abstract

This paper revisits models of typed λ-calculus based on filters of intersection types:

By using *non-idempotent* intersections, we simplify a methodology that produces modular proofs of strong normalisation based on filter models. Non-idempotent intersections provide a decreasing measure proving a key termination property, simpler than the reducibility techniques used with idempotent intersections.

Such filter models are shown to be captured by orthogonality techniques: we formalise an abstract notion of *orthogonality model* inspired by classical realisability, and express a filter model as one of its instances, along with two term-models (one of which captures a now common technique for strong normalisation).

Applying the above range of model constructions to Curry-style System F describes at different levels of detail how the *infinite polymorphism* of System F can systematically be reduced to the *finite polymorphism* of intersection types.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases Non-idempotent intersections, System F, realisability

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.51

1 Introduction

Intersection types were introduced in [9], extending the simply-typed λ-calculus with a notion of finite polymorphism. This is achieved by a new construct $A \cap B$ in the syntax of types and new typing rules such as the one on the right, where $M : A$ denotes that a term M is of type A.

$$\frac{M : A \quad M : B}{M : A \cap B}$$

One of the motivations was to characterise strongly normalising λ-terms, the property that a λ-term can be typed if and only if it is strongly normalising. Variants of intersection types systems have been studied to characterise other evaluation properties of λ-terms and served as the basis of corresponding semantics [4, 23, 31, 17, 14, 1].

In particular, intersection types were used to build filter models of λ-calculus as early as [4]. For instance, [1] reveals how the notion of intersection type filter can be tuned so that the corresponding filter models identify those λ-terms that are convertible by various restrictions of β- and η-conversion. Here we rather develop the approach of [10] showing how filters of intersection types can be used to produce models of various *source typing systems*; [10] provides a modular proof that λ-terms that are typable in some (dependent) type theory (the source system) are typable in a unique strongly normalising system of intersection types (the target system), and are therefore strongly normalising.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 51–66

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

The contributions of this paper are threefold:

A new target system.

First, we show an improvement on the methodology of [10], changing the target system of idempotent intersection types to a target system that uses *non-idempotent* intersection types (which also departs from [1]). In other words we drop the assumption $A \cap A = A$, which corresponds to the understanding of the judgement $M : A \cap B$ as follows: M can be used as data of type A or data of type B. By dropping idempotency the meaning of $M : A \cap B$ is strengthened in that M **will** be used **once** as data of type A and **once** as data of type B.

The benefit of that move is that the strong normalisation of this new target system follows from the fact that typing trees get strictly smaller with every β-reduction. This is significantly simpler than the strong normalisation of the simply-typed λ-calculus and, even more so, of its extension with idempotent intersection types (for which [10] involves reducibility techniques [18, 30]). Strangely enough there is no price to pay for this simplification, as the construction and correctness of the filter models with respect to a source system is not made harder by non-idempotency.

While this improvement concerns any of the source systems treated in [10], we choose to illustrate the methodology with a concrete source system that includes the impredicative features of System F [18], as suggested in the conclusion of [10]. As explained below, this choice is motivated by an original study of polymorphism.

We propose as a target system a variant of the system in [6], which refined with quantitative information the property that a λ-term is strongly normalising if and only if it can be typed: the length of the longest β-reduction sequence starting from a strongly normalising λ-term can be read off its typing tree. That system was inspired by the pioneering work of [20, 27] as well as [12, 13], where these ideas were connected to the tradition of resource and differential λ-calculi [8, 15] and semantics of linear logic [19]. Although from linear logic have emerged typing systems providing control over the complexity of functional programs [2, 3, 22, 16], let us emphasise that no linearity constraint is here imposed and all strongly normalising λ-terms can be typed (including non-linear ones). In this we also depart from the complexity results specific to the simply-typed λ-terms [29, 5].

Orthogonality models.

The second main contribution of this paper is to show how the above methodology can be formalised in the framework of *orthogonality*. Orthogonality underlies linear logic and its models [19] as well as classical realisability [11, 21, 26], and is used to prove properties of proofs or programs [28, 25, 24].

We formalise here a parametric model construction by introducing an abstract notion of *orthogonality model*, which we illustrate with three different instances:

- one instance based on strongly normalising λ-terms, which captures the traditional use of orthogonality to prove strong normalisation (adapted from [28, 24])
- one instance based on λ-terms that are typable with intersection types
- one instance based on filters of intersection types

To our knowledge, this is the first time that some filter models are shown to be captured by orthogonality techniques. Also, the systematic and modular approach offered by the abstract notion of orthogonality model facilitates the comparison of different proof techniques, e.g. while studying polymorphism.

Polymorphism.

The third contribution of this paper is to use the above technology to shed some new light on finite and infinite polymorphism:

System F and its extensions can assign a type $\forall\alpha\mathfrak{A}$ to a λ-term M, a form of *infinite*

polymorphism, as M can be used with any of the infinitely many instances of $\mathfrak{A}$ as its type. Terms that are typed in System F are strongly normalising [18, 30], and strongly normalising terms can be typed with intersections [9], so a direct corollary is that terms that are typed with infinite polymorphism can in fact be typed with finite polymorphism.

Our model constructions analyse this phenomenon at various levels. The finer-grained analysis is obtained from our filter models, as these give some insight on how the typing trees of System F are transformed into typing trees with finite intersections, where the useful instances of System F types have been computed. Similar ideas were investigated in [32].

Section 2 presents a target system $\lambda_\cap$ of non-idempotent intersection types and its basic properties like strong normalisation. In Section 3 we build filters of non-idempotent intersection types, showing how the use of a target system such as $\lambda_\cap$ simplifies the methodology of [10]; full details are given for concrete examples of source systems such as System F. Section 4 presents the abstract notion of orthogonality model, for the source systems already mentioned. Section 5 presents three instances, one of which captures the construction of a filter model by orthogonality techniques. In Section 6 we discuss how the infinite polymorphism of System F is reduced to the finite polymorphism of intersection types, comparing the different models we have built; then we conclude.

2 Non-idempotent intersection types, improved

Our first goal is to show how non-idempotent intersection types simplify the methodology introduced in [10]. We can use for that the system of non-idempotent intersection types of [6], yet we take in this section the opportunity to make this paper self-contained and present a more syntax-directed variant $\lambda_\cap$ that makes the proofs even simpler.

2.1 Grammar of types and properties

▶ **Definition 1** (Types). Intersection types are defined by the following syntax:

$$\begin{array}{lll} F, G, \ldots & ::= & \tau \mid A \to F \qquad F\text{-types} \\ A, B, \ldots & ::= & F \mid A \cap B \qquad A\text{-types} \\ U, V, \ldots & ::= & A \mid \omega \qquad U\text{-types} \end{array}$$

The intersection $U \cap V$ of arbitrary U-types U and V can be defined by extending the intersection of A-types with: $A \cap \omega := A$, $\omega \cap A := A$ and $\omega \cap \omega := \omega$.

Note that we do **not** assume any implicit equivalence between intersection types (such as idempotency, associativity, commutativity).

▶ Remark. For all U and V, we have $U \cap \omega = \omega \cap U$, and if $U \cap V = \omega$ then $U = V = \omega$.

▶ **Definition 2** ($\approx$). We inductively define $U \approx V$ by the following rules:

$$\frac{}{F \approx F} \qquad \frac{}{A \cap B \approx B \cap A} \qquad \frac{A \approx A' \quad B \approx B'}{A \cap B \approx A' \cap B'} \qquad \frac{A \approx B \quad B \approx C}{A \approx C}$$

$$\frac{}{(A \cap B) \cap C \approx A \cap (B \cap C)} \qquad \frac{}{A \cap (B \cap C) \approx (A \cap B) \cap C} \qquad \frac{}{\omega \approx \omega}$$

The intersection types that we use here differ from those of [6], in that the associativity and commutativity (AC) of the intersection $\cap$ are only featured "on the surface" of types,

and not underneath functional arrows $\to$. This will make the typing rules much more syntax-directed, simplifying the proofs of soundness and completeness of typing with respect to the strong normalisation property. More to the point, this approach reduces the use of the AC properties to the only places where they are needed. Interestingly enough, the idea of not using an equational theory beneath functional arrows is suggested in [1], Remark 3, but not investigated. We have the following properties (the proof can be found in [7]):

▶ **Lemma 3.** *For all U, V, W, F, U', V',*

1. *$\approx$ is an equivalence relation.*
2. *If $U \approx \omega$ then $U = \omega$ and if $U \approx F$ then $U = F$.*
3. *$U \cap V \approx V \cap U$ and $(U \cap V) \cap W \approx U \cap (V \cap W)$.*
4. *If $U \approx U'$ and $V \approx V'$ then $U \cap V \approx U' \cap V'$.*
5. *For all U and V, if $U \cap V \approx U$ then $V = \omega$.*

We equip intersection types with a notion of sub-typing:

▶ **Definition 4** ($\subseteq$)**.** We write $U \subseteq V$ if there exists U' such that $U \approx V \cap U'$.

▶ **Lemma 5.** *For all U, U', V, V' :*

1. *$\subseteq$ is a partial pre-order for intersection types, and $U \approx U'$ if and only if $U \subseteq U'$ and $U' \subseteq U$.*
2. *$U \cap V \subseteq U$ and $U \subseteq \omega$*
3. *If $U \subseteq U'$ and $V \subseteq V'$ then $U \cap V \subseteq U' \cap V'$*

2.2 Typing contexts

We now lift those concepts to typing contexts before presenting the typing rules.

▶ **Definition 6** (Contexts)**.** A context Γ is a total map from variables to U-types such that $\mathsf{Dom}(\Gamma) := \{x \mid \Gamma(x) \neq \omega\}$ is finite. The intersection of contexts $\Gamma \cap \Delta$, and the relations $\Gamma \approx \Delta$ and $\Gamma \subseteq \Delta$, are defined point-wise.

By $()$ we denote the context mapping every variable to ω and by $x{:}U$ the context mapping x to U and every other variable to ω.

The special case of $\Gamma \cap \Delta$ when $\mathsf{Dom}(\Gamma)$ and $\mathsf{Dom}(\Delta)$ are disjoint is denoted Γ, Δ.

▶ **Lemma 7** (Properties of contexts)**.** *For all contexts Γ, Γ', Δ, Δ', Γ'',*

1. *$\Gamma \cap () = \Gamma = () \cap \Gamma$ (for instance $\Gamma, x{:}\omega = \Gamma = x{:}\omega, \Gamma$)*
2. *If $\Gamma \cap \Delta = ()$ then $\Gamma = \Delta = ()$ and if $\Gamma \approx ()$ then $\Gamma = ()$*
3. *$\approx$ is an equivalence relation on contexts*
4. *$\Gamma \cap \Delta \approx \Delta \cap \Gamma$ and $(\Gamma \cap \Gamma') \cap \Gamma'' \approx \Gamma \cap (\Gamma' \cap \Gamma'')$*
5. *If $\Gamma \approx \Gamma'$ and $\Delta \approx \Delta'$ then $\Gamma \cap \Gamma' \approx \Delta \cap \Delta'$*
6. *$\Gamma \subseteq \Delta$ if and only if there exists Γ' such that $\Gamma \approx \Delta \cap \Gamma'$.*
7. *$\subseteq$ is a partial pre-order for contexts, and $\Gamma \approx \Delta$ if and only if $\Gamma \subseteq \Delta$ and $\Delta \subseteq \Gamma$.*
8. *$\Gamma \cap \Delta \subseteq \Gamma$*
9. *If $\Gamma \subseteq \Gamma'$ and $\Delta \subseteq \Delta'$ then $\Gamma \cap \Delta \subseteq \Gamma' \cap \Delta'$.*
10. *$(\Gamma, x{:}U) \subseteq \Gamma$, in particular $\Gamma \subseteq ()$.*

2.3 The target system $\lambda_\cap$

We finally present the typing rules for system $\lambda_\cap$, and its basic properties.

▶ **Definition 8** (λ-calculus)**.** Let Λ be the set of λ-terms defined by the grammar

$$M, N ::= \ x \mid \lambda x.M \mid MN$$

$\lambda x.M$ binds x in M, the free variables $fv(M)$ of a term M are defined as usual and terms are considered up to α-equivalence. The reduction rule is β-reduction:

$$(\lambda x.M)\ N \longrightarrow \ \{{}^{N}/_{x}\}M$$

The congruent closure of this rule is denoted $\longrightarrow_\beta$. SN denotes the set of strongly normalising λ-terms (for β-reduction).

$$\frac{}{x:F \vdash x:F} \qquad \frac{\Gamma, x:U \vdash M:F \quad A \subseteq U}{\Gamma \vdash \lambda x.M : A \to F}$$

$$\frac{\Gamma \vdash M:A \to F \quad \Delta \vdash N:A}{\Gamma \cap \Delta \vdash MN:F} \qquad \frac{\Gamma \vdash M:A \quad \Delta \vdash M:B}{\Gamma \cap \Delta \vdash M:A \cap B} \qquad \frac{}{\vdash M:\omega}$$

Figure 1 System $\lambda_\cap$

▶ **Definition 9** (Typability in System $\lambda_\cap$)**.** The judgement $\Gamma \vdash_\cap M:A$ denotes the derivability of $\Gamma \vdash M:A$ with the rules of Fig. 1. We write $\Gamma \vdash^n_\cap M:A$ if there exists a derivation with n uses of the application rule.

Note that the rule deriving $\vdash M:\omega$ does not interfere with the rest of the system as ω is not an A-type. It is only here for convenience to synthetically express some statements and proofs that would otherwise need a verbose case analysis (e.g. Lemma 11).

Only strongly normalising terms can be assigned an A-type by the system (Theorem 13). In fact, all of them can (Theorem 39), see for instance how the example on the side correctly uses the abstraction rule ($A \subseteq \omega$). Owing to non-idempotency, no closed term inhabits the simple type $(\tau{\to}\tau{\to}\tau'){\to}(\tau{\to}\tau')$ (with $\tau \neq \tau'$), but its natural inhabitant $\lambda f.\lambda x.f\ x\ x$ in a simply-typed system can here be given type $(\tau{\to}\tau{\to}\tau'){\to}(\tau \cap \tau{\to}\tau')$.

$$\dfrac{\dfrac{}{x:F, y:\omega \vdash \lambda y.x:F}}{x:F \vdash \lambda y.x:A{\to}F}$$

Finally, note that the introduction rule for the intersection is directed by the syntax of the type: deciding which rule instance is at the root of a derivation tree typing a term with an intersection, is entirely deterministic. We are not aware of any intersection type system featuring this property, which is here a consequence of dropping the implicit AC properties of intersections, and a clear advantage over the system in [6].

▶ **Lemma 10** (Basic properties of $\lambda_\cap$)**.**

1. *If* $\Gamma \vdash^n_\cap M:U \cap V$ *then there exist* Γ_1, Γ_2, n_1, n_2 *such that* $n = n_1 + n_2$, $\Gamma = \Gamma_1 \cap \Gamma_2$, $\Gamma_1 \vdash^{n_1}_\cap M:U$ *and* $\Gamma_2 \vdash^{n_2}_\cap M:V$.
2. *If* $\Gamma \vdash_\cap M:U$, *then* $\mathsf{Dom}(\Gamma) = fv(M)$.
3. *If* $\Gamma \vdash^n_\cap M:U$ *and* $U \approx U'$ *then there exists* Γ' *such that* $\Gamma \approx \Gamma'$ *and* $\Gamma' \vdash^n_\cap M:U'$
4. *If* $\Gamma \vdash^n_\cap M:U$ *and* $U \subseteq V$ *then there exist* m *and* Δ *such that* $m \leq n$, $\Gamma \subseteq \Delta$ *and* $\Delta \vdash^m_\cap M:V$.

2.4 Strong normalisation of $\lambda_\cap$

This is where non-idempotent intersections provide a real advantage over idempotent ones, as every β-reduction strictly reduces the number of application rules in the typing trees. The proofs, easily adapted from those in [6], can be found in [7].

▶ **Lemma 11** (Typing substitutions). *If* $\Gamma, x:U \vdash^n_\cap M:A$ *and* $\Delta \vdash^m_\cap N:U$ *then there exists* Γ' *such that* $\Gamma' \approx \Gamma \cap \Delta$ *and* $\Gamma' \vdash^{n+m}_\cap \{N/x\}M:A$.

▶ **Theorem 12** (Subject Reduction). *If* $\Gamma \vdash^n_\cap M:A$ *and* $M \longrightarrow_\beta M'$ *then there exist* m *and* Δ *such that* $m < n$, $\Gamma \subseteq \Delta$ *and* $\Delta \vdash^m_\cap M':A$.

▶ **Theorem 13** (Strong Normalisation). *If* $\Gamma \vdash_\cap M:A$ *then* $M \in \mathit{SN}$.

The converse is also true (strongly normalising terms can be typed in $\lambda_\cap$), see Theorem 39 and more generally Appendix A (with Subject Expansion, etc.).

3 Building an I-filter model for a source system

In this section we show how to use non-idempotent intersection types to simplify the methodology of [10], which we briefly review here:

The goal is to produce modular proofs of strong normalisation for various *source typing systems*. The problem is reduced to the strong normalisation of a unique *target system* of intersection types, chosen once and for all. This is done by interpreting each λ-term t as the set $[\![t]\!]$ of the intersection types that can be assigned to t in the target system. Two facts then remain to be proved:

1. if t can be typed in the source system, then $[\![t]\!]$ is not empty
2. the target system is strongly normalising

The first point is the only part that is specific to the source typing system: it amounts to turning the interpretation of terms into a filter model of the source typing system. The second point depends on the chosen target system: as [10] uses a system of *idempotent* intersection types (extending the simply-typed λ-calculus), their proof involves the usual reducibility technique [18, 30]. But this is somewhat redundant with point 1 which uses similar techniques to prove the correctness of the filter model with respect to the source system.[1]

In this paper we propose to use *non-idempotent* intersection types for the target system, so that point 2 can be proved with simpler techniques than in [10] while point 1 is not impacted by the move. In practice we propose $\lambda_\cap$ as the target system (that of [6] would work just as well).[2] We now show the details of this alternative.

3.1 I-filters of non-idempotent intersection types

The following filter constructions only involve the syntax of types and are independent from the chosen target system.

▶ **Definition 14** (I-filter).

An I-filter is a set v of A-types such that:

[1] If reducibility techniques are needed for the latter, why not use them on the source system directly (besides formulating a modular methodology)?

[2] Correspondingly, there is no simple way to embed the simply-typed λ-calculus into $\lambda_\cap$, other than considering it as the source system of the present methodology.

  - for all A and B in v we have $A \cap B \in v$
  - for all A and B, if $A \in v$ and $A \subseteq B$ then $B \in v$
- In particular the empty set and the sets of all A-types are I-filters and we write them $\bot$ and $\top$ respectively.
- Let $\mathcal{D}$ be the set of all non-empty I-filters; we call such I-filters *values*.
- Let $\mathcal{E}$ be the set of all I-filters ($\mathcal{E} = \mathcal{D} \cup \{\bot\}$).

While our intersection types differ from those in [10] (in that idempotency is dropped), the stability of a filter under type intersections makes it validate idempotency (it contains A if and only if it contains $A \cap A$, etc). This makes our filters very similar to those in [10], so we can plug-in the rest of the methodology with minimal change.

▶ Remark (Basic properties of I-filters).
1. If $(v_i)_{i\in I}$ is an non empty family of $\mathcal{E}$ then $\bigcap_{i\in I} v_i \in \mathcal{E}$.
2. If v is a set of A-types then there is a smallest $v' \in \mathcal{E}$ such that $v \subseteq v'$ and we write $< v >:= v'$.
3. If v is a set of F-types then $< v >$ is the closure of v under finite intersections.
4. If $v \in \mathcal{E}$ then $v =< \{F \mid F \in v\} >$.
5. If u and v are sets of F-types such that $< u >=< v >$ then $u = v$.

Hence, in order to prove that two I-filters are equal we just have to prove that they contain the same F-types.

I-filters form an applicative structure:

▶ **Definition 15** (Application of I-filters)**.** If u, v are in $\mathcal{E}$ then define

$$u@v := < \{F \mid \exists A \in v, (A \to F) \in u\} >$$

▶ Remark. For all $u \in \mathcal{E}$, $u@\bot = \bot@u = \bot$, and for all $u \in \mathcal{D}$, $\top@u = \top$.

▶ **Definition 16** (Environments and contexts)**.** An *environment* is a map from term variables $x, y, \ldots$ to I-filters. If ρ is an environment and Γ is a context, we say that $\Gamma \in \rho$, or Γ is *compatible with* ρ, if for all x, $\Gamma(x) = \omega$ or $\Gamma(x) \in \rho(x)$.

▶ Remark (Environments are I-filters of contexts). [3] Let ρ be an environment.
1. If $\Gamma \in \rho$ and $\Gamma' \in \rho$ then $\Gamma \cap \Gamma' \in \rho$.
2. If $\Gamma \in \rho$ and Γ' is a context such that $\Gamma \subseteq \Gamma'$ then $\Gamma' \in \rho$.

3.2 Interpretation of terms

The remaining ingredients now involve the target system; we treat here $\lambda_\cap$.

▶ **Definition 17** (Interpretation of terms)**.** If M is a term and ρ is an environment we define

$$[\![M]\!]_\rho := \{A \mid \exists \Gamma \in \rho, \Gamma \vdash_\cap M : A\}$$

▶ Remark. $[\![M]\!]_\rho \in \mathcal{E}$, and therefore $[\![M]\!]_\rho =< \{F \mid \exists \Gamma \in \rho, \Gamma \vdash_\cap M : F\} >$.

▶ **Theorem 18** (Inductive characterisation of the interpretation)**.**
1. $[\![x]\!]_\rho = \rho(x)$
2. $[\![MN]\!]_\rho = [\![M]\!]_\rho @ [\![N]\!]_\rho$

[3] Conversely, if E is an I-filter of contexts then ρ, defined by $\rho(x) = \{\Gamma(x) \neq \omega \mid \Gamma \in E\}$ for all x, is an environment.

3. $[\![\lambda x.M]\!]_\rho @ u = [\![M]\!]_{\rho, x \mapsto u}$ *for any value* u

This theorem (proved in [7]) makes $\lambda_\cap$ a suitable alternative as a target system: the filter models of the source systems treated in [10] can be done with a system of non-idempotent intersection types. While we could develop those constructions, we prefer to cover a new range of source systems: those with second-order quantifiers such as System F, as this will shed a new light on polymorphism.

3.3 Concrete examples for the source system

▶ **Definition 19** (Types and Typing System)**.** Types are built as follows:

$$\mathfrak{A}, \mathfrak{B}, \ldots ::= \ \alpha \mid \mathfrak{A} \to \mathfrak{B} \mid \mathfrak{A} \cap \mathfrak{B} \mid \forall \alpha \mathfrak{A}$$

where α denotes a type variable, $\forall \alpha \mathfrak{A}$ binds α in $\mathfrak{A}$, types are considered modulo α-conversion, and $ftv(\mathfrak{A})$ denotes the free (type) variables of $\mathfrak{A}$.

Typing contexts, denoted $\mathfrak{G}$, $\mathfrak{H}, \ldots$ are partial maps from term variables to types, $(x : \mathfrak{A})$ denotes the map from x to $\mathfrak{A}$, and . $\mathfrak{G}, \mathfrak{H}$ denotes the union of $\mathfrak{G}$ and $\mathfrak{H}$ (as graphs). Fig. 2 shows a collection of well-known typing rules to type pure λ-terms.

$$\frac{}{\mathfrak{G}, x : \mathfrak{A} \vdash x : \mathfrak{A}} \qquad \frac{\mathfrak{G}, x : \mathfrak{A} \vdash M : \mathfrak{B}}{\mathfrak{G} \vdash \lambda x.M : \mathfrak{A} \to \mathfrak{B}} \qquad \frac{\mathfrak{G} \vdash M : \mathfrak{A} \to \mathfrak{B} \quad \mathfrak{G} \vdash N : \mathfrak{A}}{\mathfrak{G} \vdash M\ N : \mathfrak{B}}$$

$$\frac{\mathfrak{G} \vdash M : \mathfrak{A} \quad \mathfrak{G} \vdash M : \mathfrak{B}}{\mathfrak{G} \vdash M : \mathfrak{A} \cap \mathfrak{B}} \qquad \frac{\mathfrak{G} \vdash M : \mathfrak{A} \cap \mathfrak{B}}{\mathfrak{G} \vdash M : \mathfrak{A}} \qquad \frac{\mathfrak{G} \vdash M : \mathfrak{A} \cap \mathfrak{B}}{\mathfrak{G} \vdash M : \mathfrak{B}}$$

$$\frac{\mathfrak{G} \vdash M : \mathfrak{A}}{\mathfrak{G} \vdash M : \forall \alpha \mathfrak{A}}\ \alpha \notin ftv(\mathfrak{G}) \qquad \frac{\mathfrak{G} \vdash M : \forall \alpha \mathfrak{A}}{\mathfrak{G} \vdash M : \{\mathfrak{B}/\alpha\}\mathfrak{A}}$$

Figure 2 Miscellaneous Typing Rules

Let $\mathcal{S}$ be the typing system consisting of an arbitrary subset of the rules. Typability in system $\mathcal{S}$ will be expressed by judgements of the form $\mathfrak{G} \vdash_{\mathcal{S}} M : \mathfrak{A}$.

We now build the model $\mathcal{M}^{\mathsf{i}}_{\mathcal{F}}$, starting with a notion of realisability candidate:

▶ **Definition 20** (Realisability Predicate)**.** A *realisability predicate* is a subset X of $\mathcal{D}$ containing $\top$. We define $\mathsf{TP}(\mathcal{D})$ as the set of realisability predicates.

▶ **Lemma 21** (Shape of realisability predicates)**.**

1. *If* $(X_i)_{i \in I}$ *an non empty family of* $\mathsf{TP}(\mathcal{D})$ *then* $\bigcap_{i \in I} X_i \in \mathsf{TP}(\mathcal{D})$.

2. *If* X *and* Y *in* $\mathsf{TP}(\mathcal{D})$ *then* $X \to Y \in \mathsf{TP}(\mathcal{D})$ *where* $X \to Y$ *is defined as*

$$X \to Y := \ \{u \mid \forall v \in X, u @ v \in Y\}$$

Proof. The only subtle point is the second one: First, for all $v \in X$, $v \neq \bot$ and thus $\top @ v = \top \in Y$. So $\top \in X \to Y$. Second, suppose that $\bot \in X \to Y$. As $X \neq \emptyset$, there is $u \in X$, for which $\bot @ u = \bot \in Y$, which contradicts $Y \in \mathsf{TP}(\mathcal{D})$. ◀

The model $\mathcal{M}^{\mathsf{i}}_{\mathcal{F}}$ consists of the interpretations of terms (Definition 17) and types:

▶ **Definition 22** (Interpretation of types).
Valuations are mappings from type variables to elements of $\mathsf{TP}(\mathcal{D})$.

Given such a valuation σ, the interpretation of types is defined as follows:

$$\begin{array}{lll} [\![\alpha]\!]_\sigma & := & \sigma(\alpha) \\ [\![\mathfrak{A}{\to}\mathfrak{B}]\!]_\sigma & := & [\![\mathfrak{A}]\!]_\sigma {\to} [\![\mathfrak{B}]\!]_\sigma \end{array} \qquad \begin{array}{lll} [\![\mathfrak{A}\cap\mathfrak{B}]\!]_\sigma & := & [\![\mathfrak{A}]\!]_\sigma \cap [\![\mathfrak{B}]\!]_\sigma \\ [\![\forall\alpha\mathfrak{A}]\!]_\sigma & := & \bigcap_{X\in\mathsf{TP}(\mathcal{D})} [\![\mathfrak{A}]\!]_{\sigma,\alpha\mapsto X} \end{array}$$

The interpretation of typing contexts is defined as follows:

$$[\![\mathfrak{G}]\!]_\sigma := \{\rho \mid \forall (x{:}\mathfrak{A}) \in \mathfrak{G}, \rho(x) \in [\![\mathfrak{A}]\!]_\sigma\}$$

Finally we get Adequacy, by a simple induction on derivations, using Theorem 18:

▶ **Lemma 23** (Adequacy Lemma). *If $\mathfrak{G} \vdash_{\mathcal{S}} M : \mathfrak{A}$, then for all valuations σ and for all mappings $\rho \in [\![\mathfrak{G}]\!]_\sigma$ we have $[\![M]\!]_\rho \in [\![\mathfrak{A}]\!]_\sigma$.*

▶ **Corollary 24** (Strong normalisation of $\mathcal{S}$). *If $\mathfrak{G} \vdash_{\mathcal{S}} M{:}\mathfrak{A}$ then $M \in \mathsf{SN}$.*

Proof. Applying the previous lemma with σ mapping every type variable to $\{\top\}$ and ρ mapping all term variable to $\top$, we get $[\![M]\!]_\rho \in [\![\mathfrak{A}]\!]_\sigma$, so $[\![M]\!]_\rho \neq \bot$. Hence, M can be typed in $\lambda_\cap$, so $M \in \mathsf{SN}$. ◀

The advantage of non-idempotent intersection types (over idempotent ones) lies in the very last step of the above proof: here the typing trees of $\lambda_\cap$ get smaller with every β-reduction (proof of Theorem 13), while a reducibility technique as in [10] combines yet again an induction on types with an induction on typing trees similar to that in the Adequacy Lemma.

4 Orthogonality models of typed λ-calculi

In this section we show how the above methodology can be integrated to the theory of *orthogonality*, i.e. how this kind of filter model construction can be captured by orthogonality techniques [19, 11, 21, 26]. These techniques are particularly suitable to prove that typed terms satisfy some property [28, 25, 24], the most well-known of which being Strong Normalisation.

For this we define an abstract notion of orthogonality model for a system $\mathcal{S}$ built from the rules of Fig. 2. Our definition thus applies to the simply-typed λ-calculus, the idempotent intersection type system, System F, etc but we could also adapt it with no difficulty to accommodate System F_ω.

Orthogonality techniques and the filter model construction from Section 3 (with the sets $\mathcal{D}$ and $\mathcal{E}$) inspire the notion of orthogonality model below. First we need notations:

▶ Notation. Given a set $\mathcal{D}$, let $\mathcal{D}^*$ be the set of lists of elements of $\mathcal{D}$, with [] representing the empty list and $u :: \overrightarrow{v}$ representing the list of head u and tail $\overrightarrow{v}$.

▶ **Definition 25** (Orthogonality model).
An *orthogonality model* is a 4-tuple $(\mathcal{E}, \mathcal{D}, \perp\!\!\!\perp, [\![\_]\!]\_)$ where

- $\mathcal{E}$ is a set, called the *support*
- $\mathcal{D} \subseteq \mathcal{E}$ is a set of elements called *values*
- $\perp\!\!\!\perp \subseteq \mathcal{D} \times \mathcal{D}^*$ is called the *orthogonality relation*
- $[\![\_]\!]\_$ is a function mapping every λ-term M (typed or untyped) to an element $[\![M]\!]_\rho$ of the support, where ρ is a parameter called *environment* mapping term variables to values.
- the following axioms are satisfied:

(A1) For all ρ, $\overrightarrow{v}$, x, if $\rho(x) \perp\!\!\!\perp \overrightarrow{v}$ then $[\![x]\!]_\rho \perp\!\!\!\perp \overrightarrow{v}$.

(A2) For all ρ, $\overrightarrow{v}$, M_1, M_2, if $[\![M_1]\!]_\rho \perp\!\!\!\perp ([\![M_2]\!]_\rho :: \overrightarrow{v})$ then $[\![M_1\ M_2]\!]_\rho \perp\!\!\!\perp \overrightarrow{v}$.
(A3) For all ρ, $\overrightarrow{v}$, x, M and for all values u, if $[\![M]\!]_{\rho,x\mapsto u} \perp\!\!\!\perp \overrightarrow{v}$ then $[\![\lambda x.M]\!]_\rho \perp\!\!\!\perp (u :: \overrightarrow{v})$.

In fact, $\mathcal{D}$ and $\perp\!\!\!\perp$ are already sufficient to interpret any type A as a set $[\![A]\!]$ of values (see Definition 28 below): if types are seen as logical formulae, we can see this construction as a way of building some of their realisability / set-theoretical models.

There is no notion of computation pertaining to values, but the interplay between the interpretation of terms and the orthogonality relation is imposed by the axioms so that the Adequacy Lemma (which relates typing to semantics) holds:

$$\text{If } \vdash M : A \text{ then } [\![M]\!] \in [\![A]\!]$$

4.1 Interpretation of types and Adequacy Lemma

▶ **Definition 26** (Orthogonal).
- If $X \subseteq \mathcal{D}$ then let $X^\perp := \{\overrightarrow{v} \in \mathcal{D}^* \mid \forall u \in X, u \perp\!\!\!\perp \overrightarrow{v}\}$
- If $Y \subseteq \mathcal{D}^*$ then let $Y^\perp := \{u \in \mathcal{D} \mid \forall \overrightarrow{v} \in Y, u \perp\!\!\!\perp \overrightarrow{v}\}$

▶ Remark. If $X \subseteq \mathcal{D}$ or $X \subseteq \mathcal{D}^*$ then $X \subseteq X^{\perp\perp}$ and $X^{\perp\perp\perp} = X^\perp$.

▶ **Definition 27** (Lists and Cons construct). If $X \subseteq \mathcal{D}$ and $Y \subseteq \mathcal{D}^*$, then define

$$X :: Y := \{u :: \overrightarrow{v} \mid u \in X, \overrightarrow{v} \in Y\}$$

▶ **Definition 28** (Interpretation of types).
Mappings from type variables to subsets of $\mathcal{D}^*$ are called *valuations*.

Given such a valuation σ, the interpretation of types is defined as follows:

$$\begin{array}{llll} [\alpha]_\sigma & := \sigma(\alpha) & [\mathfrak{A} \cap \mathfrak{B}]_\sigma & := [\mathfrak{A}]_\sigma \cup [\mathfrak{B}]_\sigma \\ [\mathfrak{A}\to\mathfrak{B}]_\sigma & := [\![\mathfrak{A}]\!]_\sigma :: [\mathfrak{B}]_\sigma & [\forall\alpha\mathfrak{A}]_\sigma & := \bigcup_{Y\subseteq\mathcal{D}^*} [\mathfrak{A}]_{\sigma,\alpha\mapsto Y} \end{array}$$
$$[\![\mathfrak{A}]\!]_\sigma := [\mathfrak{A}]_\sigma^\perp$$

The interpretation of typing contexts is defined as follows:

$$[\![\mathfrak{G}]\!]_\sigma := \{\rho \mid \forall (x:\mathfrak{A}) \in \mathfrak{G}, \rho(x) \in [\![\mathfrak{A}]\!]_\sigma\}$$

▶ Remark. Note that $[\{\mathfrak{B}/\alpha\}\mathfrak{A}]_\sigma = [\mathfrak{A}]_{\sigma,\alpha\mapsto[\mathfrak{B}]_\sigma}$ and $[\![\{\mathfrak{B}/\alpha\}\mathfrak{A}]\!]_\sigma = [\![\mathfrak{A}]\!]_{\sigma,\alpha\mapsto[\mathfrak{B}]_\sigma}$.
Also note that $[\![\mathfrak{A}\cap\mathfrak{B}]\!]_\sigma = [\![\mathfrak{A}]\!]_\sigma \cap [\![\mathfrak{B}]\!]_\sigma$ and $[\![\forall\alpha\mathfrak{A}]\!]_\sigma = \bigcap_{Y\subseteq\mathcal{D}^*} [\![\mathfrak{A}]\!]_{\sigma,\alpha\mapsto Y}$.

An orthogonality model is a sufficiently rich structure for Adequacy to hold:

▶ **Lemma 29** (Adequacy Lemma). *If $\mathfrak{G} \vdash_{\mathcal{S}} M : \mathfrak{A}$, then for all valuations σ and for all mappings $\rho \in [\![\mathfrak{G}]\!]_\sigma$ we have $[\![M]\!]_\rho \in [\![\mathfrak{A}]\!]_\sigma$.*

Proof. By induction on $\mathfrak{G} \vdash_{\mathcal{S}} M : \mathfrak{A}$, using axioms (A1), (A2) and (A3). See [7]. ◀

4.2 The special case of applicative structures

In the next section we present instances of orthogonality models. They will have in common that $\mathcal{E}$ is an applicative structure, as we have seen with I-filters. This motivates the following notion:

▶ **Definition 30** (Applicative orthogonality model).
An *applicative orthogonality model* is a 4-tuple $(\mathcal{E}, \mathcal{D}, @, [\![\_]\!]\_)$ where:

- $\mathcal{E}$ is a set, $\mathcal{D}$ is a subset of $\mathcal{E}$, @ is a (total) function from $\mathcal{E} \times \mathcal{E}$ to $\mathcal{E}$, and $[\![\_]\!]\_$ is a function (parameterised by an environment) from λ-terms to the support.
- $(\mathcal{E}, \mathcal{D}, \perp\!\!\!\perp, [\![\_]\!]\_)$ is an orthogonality model,
  where the relation $u \perp\!\!\!\perp \overrightarrow{v}$ is defined as $(u@\overrightarrow{v}) \in \mathcal{D}$
  (writing $u@\overrightarrow{v}$ for $(\dots(u@v_1)@\dots@v_n)$ if $\overrightarrow{v} = v_1 :: \dots v_n :: []$).

▶ Remark. Axioms (A1) and (A2) are ensured provided that $[\![M\ N]\!]_\rho = [\![M]\!]_\rho @ [\![N]\!]_\rho$ and $[\![x]\!]_\rho = \rho(x)$. These conditions can hold by definition (as in term models, cf. the next Section), or can be proved (as in Theorem 18, which also proves (A3)).

5 Three instances of orthogonality models

We now give three instances of (applicative) orthogonality models with well-chosen sets of values, applications, and interpretations of terms, so that, from $[\![M]\!] \in [\![A]\!]$, we can eventually derive the strong normalisation of the λ-term M.

The first two instances are term models: Terms are interpreted as themselves (see Definition 31), so the support is the set of all λ-terms seen as an applicative structure (using term application: $M_1 @^{\text{TERM}} M_2 := M_1\ M_2$).

▶ **Definition 31** (Interpretation of terms in a term model).

$$\begin{array}{lcl} [\![x]\!]_\rho^{\text{TERM}} & := & \rho(x) \\ [\![M_1\ M_2]\!]_\rho^{\text{TERM}} & := & [\![M_1]\!]_\rho^{\text{TERM}}\ [\![M_2]\!]_\rho^{\text{TERM}} \\ [\![\lambda x.M]\!]_\rho^{\text{TERM}} & := & \lambda x.[\![M]\!]_{\rho, x\mapsto x}^{\text{TERM}} \end{array}$$

▶ Remark. $[\![\{N/x\} M]\!]_\rho^{\text{TERM}} = [\![M]\!]_{\rho, x\mapsto [\![N]\!]_\rho^{\text{TERM}}}^{\text{TERM}}$

In the first instance, values are strongly normalising terms themselves; that instance rephrases, with an orthogonality model, standard proofs of strong normalisation by orthogonality or reducibility candidates [28, 24]. In the second instance, values are the terms that can be typed with intersection types, for instance in system $\lambda_\cap$.

The third instance is not a term model but a syntax-free model, where a term is interpreted as the filter of the intersection types that it can be assigned (e.g. in $\lambda_\cap$, see Definition 17), and orthogonality is defined in terms of filters being non-empty.

In the second and third instances, Strong Normalisation follows from the fact that terms typable with intersection types are themselves strongly normalising (Theorem 13 for $\lambda_\cap$).

▶ **Theorem 32.** *The structures*

- $\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}} = (\Lambda, \mathsf{SN}, @^{\text{TERM}}, [\![\_]\!]^{\text{TERM}})$
- $\mathcal{M}^{\perp\!\!\!\perp}_{\cap} = (\Lambda, \Lambda_\cap, @^{\text{TERM}}, [\![\_]\!]^{\text{TERM}}_{\_})$ *(where $\Lambda_\cap$ is the set of λ-terms typable in $\lambda_\cap$)*
- $\mathcal{M}^{\perp\!\!\!\perp}_{\mathcal{F}} = (\mathcal{E}, \mathcal{D}, @, [\![\_]\!]\_)$ *(with the four components as defined in Section 3)*

are applicative orthogonality models.

Indeed, as mentioned in Section 4.2, the applicative structures $\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}}$ and $\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$ already satisfy axioms (A1) and (A2) because of Definition 31. The structure $\mathcal{M}^{\perp\!\!\!\perp}_{\mathcal{F}}$ satisfies them, as well as axiom (A3), as immediate consequences of Theorem 18. Axiom (A3) holds in $\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}}$ and $\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$ because of their respective expansion properties:

▶ **Lemma 33** (Expansion).

1. *If $\{P/x\} M\ \overrightarrow{N} \in \mathsf{SN}$ and $P \in \mathsf{SN}$ then $(\lambda x.M)\ P\ \overrightarrow{N} \in \mathsf{SN}$.*
2. *If $\{P/x\} M\ \overrightarrow{N} \in \Lambda_\cap$ and $P \in \Lambda_\cap$ then $(\lambda x.M)\ P\ \overrightarrow{N} \in \Lambda_\cap$.*

Admittedly, once $\lambda_\cap$ has been proved to characterise SN (Theorems 13 and 39), the two points are identical and so are the two models $\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}}$ and $\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$. But involving the bridge of

this characterisation goes much beyond than needed for either point: point 1 is a known fact of the literature; point 2 is a simple instance of Subject Expansion (Theorem 38, Appendix A) not requiring Subject Reduction (Theorem 12) while both are involved at some point in the more advanced property $\mathsf{SN} = \Lambda_\cap$. In brief, as we are interested in comparing proof *techniques* for the strong normalisation of System $\mathcal{S}$, the question of which properties are used and in which order matters.

▶ Remark. In both structures $\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}}$ and $\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$ we can check that:
For all $\overrightarrow{N} \in \mathsf{SN}^*$ and any term variable x, $x \perp\!\!\!\perp \overrightarrow{N}$.
Hence, for all valuations σ and all types $\mathfrak{A}$, $x \in [\![\mathfrak{A}]\!]_\sigma$.
For $\mathcal{M}^{\perp\!\!\!\perp}_{\mathcal{F}}$ we have instead: For all list of values $\overrightarrow{v}$, $\top \perp\!\!\!\perp \overrightarrow{v}$.
Hence, for all valuations σ and all types $\mathfrak{A}$, $\top \in [\![\mathfrak{A}]\!]_\sigma$.

Now using the Adequacy Lemma (Lemma 29), we finally get:

▶ **Corollary 34.** *If $\mathfrak{G} \vdash_{\mathcal{S}} M : \mathfrak{A}$, then:*

$\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}}$ *For all valuations σ and all mappings $\rho \in [\![\mathfrak{G}]\!]_\sigma$ we have $[\![M]\!]^{\mathrm{TERM}}_\rho \in \mathsf{SN}$.*
Hence, $M \in \mathsf{SN}$.

$\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$ *For all valuations σ and all mappings $\rho \in [\![\mathfrak{G}]\!]_\sigma$*
there exist Γ and A such that $\Gamma \vdash_\cap [\![M]\!]^{\mathrm{TERM}}_\rho : A$. Hence, $M \in \mathsf{SN}$.

$\mathcal{M}^{\perp\!\!\!\perp}_{\mathcal{F}}$ *For all valuations σ and all mappings $\rho \in [\![\mathfrak{G}]\!]_\sigma$ we have $[\![M]\!]_\rho \neq \bot$.*
Hence, there exist Γ and A such that $\Gamma \vdash_\cap M : A$, and finally $M \in \mathsf{SN}$.

Proof.

$\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}}$ The second statement is obtained by choosing σ to map every type variable to the empty set, and ρ to map every term variable to itself.

$\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$ The second statement is obtained by choosing σ to map every type variable to the empty set, and ρ to map every term variable to itself; then we conclude using Theorem 13.

$\mathcal{M}^{\perp\!\!\!\perp}_{\mathcal{F}}$ The first statement holds because $\bot \notin \mathcal{D}$ and $[\![\mathfrak{A}]\!]_\sigma \subseteq \mathcal{D}$. To prove the second, we need to show that there exist such a σ and such a ρ; take σ to map every type variable to the empty set and take ρ to map every term variable to $\top$. The final result comes from Theorem 13.

◀

6 A new light on polymorphism

System $\lambda_\cap$ is a convenient call on the journey from typability in $\mathcal{S}$ to strong normalisation, as it divides the path into two parts that are different in nature: first, proving that being typable in some system implies being typable in another (a result on the transformation of typing trees); second, proving that being typable in the latter system implies strong normalisation, which is trivial in the case of $\lambda_\cap$ (for each reduction makes the typing tree smaller).

The first part of the journey is described by the Adequacy Lemma in the filter models $\mathcal{M}^{\perp\!\!\!\perp}_{\mathcal{F}}$ and $\mathcal{M}^{\mathrm{i}}_{\mathcal{F}}$, as well as in the term model $\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$: It shows that being typable in System $\mathcal{S}$ implies being typable in System $\lambda_\cap$. This is interesting in itself when applied to System F, as it sheds an interesting light on polymorphism:

As mentioned in the introduction, terms that are typed in System F, i.e. with *infinite* polymorphism, are strongly normalising (cf. model $\mathcal{M}^{\perp\!\!\!\perp}_{\mathsf{SN}}$), and can therefore be typed with intersection types (see e.g. Theorem 39), i.e. with *finite* polymorphism.

The cut or detour in the above proof can be eliminated, and this paper shows the resulting proof as the construction of another orthogonality model: namely, the term-model $\mathcal{M}^{\perp\!\!\!\perp}_{\cap}$,

where no mention is made of the strong normalisation property. The model construction is purely about the transformation of typing trees and appears to disregard normalisation properties. Yet it makes no explicit connection between the type of a λ-term in System F and the type that it gets in $\lambda_\cap$ via this method. The filter models $\mathcal{M}^{\Vdash}_{\mathcal{F}}$ and $\mathcal{M}^{\mathsf{i}}_{\mathcal{F}}$ address this issue. Indeed, a System F type $\mathfrak{A}$ (say a closed one) is interpreted as a collection $[\![\mathfrak{A}]\!]$ of filters of intersection types; computing this can be done before inhabitants of types are even considered. Each of the filters in the collection represents the exact set of intersection types that a λ-term of type $\mathfrak{A}$ (in System F) could potentially get in $\lambda_\cap$.

▶ **Example 35.** For instance in $\mathcal{M}^{\mathsf{i}}_{\mathcal{F}}$,

1. $\forall\alpha\alpha$ is interpreted as $[\![\forall\alpha\alpha]\!] = \bigcap_{X\in TP(\mathcal{D})} X = \{\top\}$.
   This means that a λ-term of type $\forall\alpha\alpha$ can necessarily be given any intersection type ($\top$ is the set of all intersection types).
2. $(\forall\alpha\alpha)\to(\forall\beta\beta)$ is interpreted as

$$\{\top\}\to\{\top\} = \{u\in\mathcal{D} \mid u@\top = \top\} = \{u\in\mathcal{D} \mid \forall F, \exists A, A\to F\in u\}$$

   Such a filter u contains an arrow towards each F-type.
3. $\forall\alpha\alpha\to\alpha$ is interpreted as $\{u\in\mathcal{D} \mid \forall X\in TP(\mathcal{D}), \forall v\in X, u@v\in X\}$.
   Such a filter u contains, for all F-types F, a type of the form $(F\cap\ldots\cap F)\to F$ (take $v =< F >$ and $X = \{v,\top\}$).

In brief, this interpretation of System F types transforms infinite polymorphism into finite polymorphism, generating collections of potential instances. The Adequacy Lemma then proves that those instances are sufficient to type the λ-terms.

In [32], a preorder with greatest lower bounds is identified in the syntax of System F types, into which intersection types can therefore be embedded. The converse is studied by a form of surjectivity of that embedding, but the example of $\vdash_F \lambda x.x\ \ldots\ x : (\forall\alpha\alpha)\to(\forall\beta\beta)$ is used to show that no intersection type exists that can type all the λ-terms of type $(\forall\alpha\alpha)\to(\forall\beta\beta)$. As we interpret System F types by *collections* of filters, different filters can be used for different terms.

While further work should relate our filters to inverse forms of the aforementioned embedding, the comparison with [32] also raises the question of whether or not the different versions of the proof that typable in System $\mathcal{S}$ implies typable in System $\lambda_\cap$, avoid the computation of the λ-terms down to their normal forms. This computation is of course present in the proof with the cut (the construction of model $\mathcal{M}^{\Vdash}_{\mathsf{SN}}$ combined with Theorem 39) and also present in [32] (type-preservation is proved by an induction on longest β-reduction sequences). One could argue that this computation is still present (yet hidden) in the construction of model $\mathcal{M}^{\Vdash}_{\cap}$ as it relies on the Subject Expansion property for $\lambda_\cap$ (Theorem 38).[4] It seems it is **not** the case for $\mathcal{M}^{\Vdash}_{\mathcal{F}}$ and $\mathcal{M}^{\mathsf{i}}_{\mathcal{F}}$: Indeed, while Adequacy does rely on the property that $[\![M]\!]_{\rho,x\mapsto[\![N]\!]_\rho}@\overrightarrow{v}\neq\bot$ implies $[\![(\lambda x.M)\ N]\!]_\rho@\overrightarrow{v}\neq\bot$, this property is **not** obtained by computing $\left\{{}^{N}\!/_{x}\right\}M$ but rather by analysing the typing trees of M and N separately.[5] This contrasts with the term models $\mathcal{M}^{\Vdash}_{\mathsf{SN}}$ and $\mathcal{M}^{\Vdash}_{\cap}$, where $\left\{{}^{N}\!/_{x}\right\}M$ is computed in the process and can generate new redexes to be further analysed.

Of course one could argue that, the typing trees in $\lambda_\cap$ of a λ-term M being bigger than the longest β-reduction sequence starting from M, producing one of them (be it with the

[4] It probably uses it as many times as there are reduction steps from a term to its normal form.
[5] The property $[\![M]\!]_{\rho,x\mapsto[\![N]\!]_\rho} = [\![\left\{{}^{N}\!/_{x}\right\}M]\!]_\rho$ holds, but not by definition.

Adequacy Lemma) is at least as hard as computing the term to its normal form. Yet $\mathcal{M}^{\Vdash}_{\mathcal{F}}$ and $\mathcal{M}^{\mathsf{i}}_{\mathcal{F}}$ could well have been defined with a traditional idempotent intersection type system [9] (in which typing trees are not correlated to the lengths of β-reduction sequences), adapting directly [10] to System F.[6]

7 Conclusion

We have seen how the use of non-idempotent intersection types simplifies the methodology from [10] by cutting a second use of reducibility techniques to prove strong normalisation. We have seen how the corresponding filter model construction can be done by orthogonality techniques, with an abstract notion of orthogonality model. As illustrated in Section 5, this notion allows a lot of work (e.g. proving the Adequacy Lemma) to be factorised, while building models like $\mathcal{M}^{\Vdash}_{\mathsf{SN}}$, $\mathcal{M}^{\Vdash}_{\cap}$ and $\mathcal{M}^{\Vdash}_{\mathcal{F}}$. Note that, while $\mathcal{M}^{\Vdash}_{\mathcal{F}}$ and $\mathcal{M}^{\mathsf{i}}_{\mathcal{F}}$ share the same ingredients $\mathcal{E}$, $\mathcal{D}$, @ and $[\![\_]\!]\_$, they are different in the way types are interpreted; see the discussion in [7].

We have also compared the models in the way they enlighten the transformation of infinite polymorphism into finite polymorphism, although more examples should be computed to illustrate and better understand the theoretical result. An objective could be to identify (and eliminate), in the interpretation of a type from System F, those filters that are not the interpretation of any term of that type. What could help this, is to force filters to be stable under type instantiation, in the view that interpretations of terms are generated by a single F-type, i.e. a *principal* type.

Proving Subject Expansion as generally as possible led us to identify a sub-reduction of β which also helps understanding how and when the semantics $[\![\_]\!]\_$ of terms is preserved, see Appendix A. This is similar to [1], and future work should adapt their methodology to accommodate our non-idempotent intersections.

Finally, as non-idempotent intersection types were used in [6] to measure the complexity of strongly normalising term, we would like to see whether we can adapt our model constructions to lift such results to the source typing systems. The hope would be to recover for instance results that are known for the simply-typed calculus [29, 5], with a methodology that can be adapted to other source systems such as System F.

References

1 Fabio Alessi, Franco Barbanera, and Mariangiola Dezani-Ciancaglini. Intersection types and lambda models. *Theoret. Comput. Sci.*, 355(2):108–126, 2006.

2 Patrick Baillot. Checking polynomial time complexity with types. In Ricardo A. Baeza-Yates, Ugo Montanari, and Nicola Santoro, editors, *IFIP TCS*, volume 223 of *IFIP Conference Proceedings*, pages 370–382. Kluwer, August 2002.

3 Patrick Baillot and Virgile Mogbil. Soft lambda-calculus: a language for polynomial time computation. *CoRR*, cs.LO/0312015, 2003.

4 Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *J. of Symbolic Logic*, 48(4):931–940, 1983.

5 Arnold Beckmann. Exact bounds for lengths of reductions in typed lambda-calculus. *J. of Symbolic Logic*, 66(3):1277–1285, 2001.

6 Alexis Bernadet and Stéphane Lengrand. Complexity of strongly normalising λ-terms via non-idempotent intersection types. In Martin Hofmann, editor, *Proc. of the 14th Int. Conf.*

[6] Non-idempotency is never used (but for the fact that it directly implies strong normalisation).

on Foundations of Software Science and Computation Structures (FOSSACS'11), volume 6604 of *LNCS*. Springer-Verlag, March 2011.

7 Alexis Bernadet and Stéphane Lengrand. Non-idempotent intersection types, orthogonality and filter models - long version. Technical report, LIX, CNRS-INRIA-Ecole Polytechnique, June 2011.

8 Gérard Boudol, Pierre-Louis Curien, and Carolina Lavatelli. A semantics for lambda calculi with resources. *Math. Structures in Comput. Sci.*, 9(4):437–482, 1999.

9 M. Coppo and M. Dezani-Ciancaglini. A new type assignment for lambda-terms. *Archiv für mathematische Logik und Grundlagenforschung*, 19:139–156, 1978.

10 Thierry Coquand and Arnaud Spiwack. A proof of strong normalisation using domain theory. *Logic. Methods Comput. Science*, 3(4), 2007.

11 Vincent Danos and Jean-Louis Krivine. Disjunctive tautologies as synchronisation schemes. In Peter Clote and Helmut Schwichtenberg, editors, *Proc. of the 9th Annual Conf. of the European Association for Computer Science Logic (CSL'00)*, volume 1862 of *LNCS*, pages 292–301. Springer-Verlag, August 2000.

12 Daniel de Carvalho. Intersection types for light affine lambda calculus. *ENTCS*, 136:133–152, 2005.

13 Daniel de Carvalho. Execution time of lambda-terms via denotational semantics and intersection types. *CoRR*, abs/0905.4251, 2009.

14 M. Dezani-Ciancaglini, F. Honsell, and Y. Motohama. Compositional characterizations of lambda-terms using intersection types (extended abstract). In *MFCS: Symp. on Mathematical Foundations of Computer Science*, 2000.

15 Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoret. Comput. Sci.*, 309(1-3):1–41, 2003.

16 Marco Gaboardi and Simona Ronchi Della Rocca. A soft type assignment system for *lambda* -calculus. In Jacques Duparc and Thomas A. Henzinger, editors, *Proc. of the 16th Annual Conf. of the European Association for Computer Science Logic (CSL'07)*, volume 4646 of *LNCS*, pages 253–267. Springer-Verlag, September 2007.

17 S. Ghilezan. Strong normalization and typability with intersection types. *Notre Dame J. Formal Loigc*, 37(1):44–52, 1996.

18 J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Thèse d'état, Université Paris 7, 1972.

19 Jean-Yves Girard. Linear logic. *Theoret. Comput. Sci.*, 50(1):1–101, 1987.

20 A. J. Kfoury and J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In *Proc. of the 26th Annual ACM Symp. on Principles of Programming Languages (POPL'99)*, pages 161–174. ACM Press, 1999.

21 Jean-Louis Krivine. Typed lambda-calculus in classical Zermelo-Frænkel set theory. *Arch. Math. Log.*, 40(3):189–205, 2001.

22 Yves Lafont. Soft linear logic and polynomial time. *Theoret. Comput. Sci.*, 318(1-2):163–180, 2004.

23 D. Leivant. Typing and computational properties of lambda expressions. *Theoret. Comput. Sci.*, 44(1):51–68, 1986.

24 Stéphane Lengrand and Alexandre Miquel. Classical F_ω, orthogonality and symmetric candidates. *Ann. Pure Appl. Logic*, 153:3–20, March 2008.

25 P.-A. Melliès and J. Vouillon. Recursive polymorphic types and parametricity in an operational framework. In Prakash Panangaden, editor, *20th Annual IEEE Symp. on Logic in Computer Science*, pages 82–91. IEEE Computer Society Press, June 2005.

26 Guillaume Munch-Maccagnoni. Focalisation and classical realisability. In Erich Grädel and Reinhard Kahle, editors, *Proc. of the 18th Annual Conf. of the European Association for*

Computer Science Logic (CSL'09), volume 5771 of *LNCS*, pages 409–423. Springer-Verlag, September 2009.

27 Peter Møller Neergaard and Harry G. Mairson. Types, potency, and idempotency: why nonlinearity and amnesia make a type system work. In Chris Okasaki and Kathleen Fisher, editors, *Proc. of the ACM Intern. Conf. on Functional Programming*, pages 138–149. ACM Press, September 2004.

28 Michel Parigot. Proofs of strong normalisation for second order classical natural deduction. *J. of Symbolic Logic*, 62(4):1461–1479, 1997.

29 Helmut Schwichtenberg. Complexity of normalization in the pure typed lambda calculus. In A. S. Troelstra and D. Van Dalen, editors, *The L. E. J. Brouwer Centenary Symposium*, Amsterdam, 1982. North-Holland.

30 W. W. Tait. A realizability interpretation of the theory of species. In *Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 240–251. Springer-Verlag, 1975.

31 S. van Bakel. Intersection Type Assignment Systems. *Theoret. Comput. Sci.*, 151(2):385–435, 1995.

32 Hirofumi Yokouchi. Embedding a second order type system into an intersection type system. *Inform. and Comput.*, 117(2):206–220, 1995.

A Completeness and preservation of semantics

We obtain completeness by identifying a reduction strategy $\hookrightarrow$ (if a term can be β-reduced then one of its sub-terms can be reduced by $\hookrightarrow$). Proofs can be found in [7].

▶ **Definition 36** ($\hookrightarrow$)**.** We define the reduction $M \hookrightarrow_\Omega M'$, where Ω is either a term or ϵ (a dummy placeholder for which $fv(\epsilon) = \emptyset$) as follows:

$$\frac{x \in fv(M)}{(\lambda x.M)N \hookrightarrow_\epsilon \left\{{}^{N}\!/_{x}\right\}M} \qquad \frac{x \notin fv(M)}{(\lambda x.M)N \hookrightarrow_N M}$$

$$\frac{M_1 \hookrightarrow_\Omega M_1'}{M_1M_2 \hookrightarrow_\Omega M_1'M_2} \qquad \frac{M_2 \hookrightarrow_\Omega M_2'}{M_1M_2 \hookrightarrow_\Omega M_1M_2'} \qquad \frac{M \hookrightarrow_\Omega M' \quad x \notin fv(\Omega)}{\lambda x.M \hookrightarrow_\Omega \lambda x.M'}$$

▶ **Lemma 37** (Typing substitutions)**.** *If* $\Gamma \vdash_\cap \left\{{}^{N}\!/_{x}\right\}M : A$ *then there exists* Γ_1, Γ_2 *and* U *such that* $\Gamma \approx \Gamma_1 \cap \Gamma_2$, $\Gamma_1, x{:}U \vdash_\cap M{:}A$ *and* $\Gamma_2 \vdash_\cap N{:}U$.

▶ **Theorem 38** (Subject Expansion)**.** *Assume* $\Gamma' \vdash_\cap M'{:}A$ *and* $M \hookrightarrow_\Omega M'$.
Assume $\Delta \vdash_\cap N{:}B$ *if* $\Omega = N$, *otherwise let* $\Delta = ()$ *when* $\Omega = \epsilon$.
Then there exists Γ *such that* $\Gamma \approx \Gamma' \cap \Delta$ *and* $\Gamma \vdash_\cap M{:}A$.

Proof. First by induction on the derivation $M \hookrightarrow_\Omega M'$, then by induction on A, using Lemma 37 for the base case. ◀

▶ **Theorem 39** (Completeness)**.** *If* $M \in \mathsf{SN}$ *there exist* Γ *and* A *such that* $\Gamma \vdash_\cap M{:}A$.

▶ **Corollary 40.** *If* $M \hookrightarrow_\Omega M'$ *and* $M' \in \mathsf{SN}$ *and* $\Omega \in \mathsf{SN} \cup \{\epsilon\}$ *then* $M \in \mathsf{SN}$.

Preservation of term interpretation under reduction can also be described in terms of $\hookrightarrow$:

▶ **Theorem 41.** *1. If* $M \longrightarrow_\beta M'$ *then for all* ρ, $[\![M]\!]_\rho \subseteq [\![M']\!]_\rho$.
2. If $M \hookrightarrow_\epsilon M'$ *then* $[\![M]\!]_\rho = [\![M']\!]_\rho$.
3. If $M \hookrightarrow_N M'$ *and* $[\![N]\!]_\rho \neq \bot$, *then* $[\![M]\!]_\rho = [\![M']\!]_\rho$.
4. If $M \hookrightarrow_N M'$ *and* $[\![N]\!]_\rho = \bot$, *then* $[\![M]\!]_\rho = \bot$.
But there are cases where $M \longrightarrow_\beta M'$ and $[\![M]\!]_\rho \neq [\![M']\!]_\rho$ (even with $[\![M]\!]_\rho \neq \bot$).
Take $M := (\lambda z.(\lambda y.a)(zz))$ and $M' := \lambda z.a$.

Algebraic characterization of FO for scattered linear orderings*

Alexis Bès[1] and Olivier Carton[2]

1 **LACL, Université Paris-Est, Créteil, France**
bes@u-pec.fr

2 **LIAFA, Université Paris Diderot Paris, France**
Olivier.Carton@liafa.jussieu.fr

Abstract

We prove that for the class of sets of words indexed by countable scattered linear orderings, there is an equivalence between definability in first-order logic, star-free expressions with marked product, and recognizability by finite aperiodic semigroups which satisfy the equation $x^\omega x^{-\omega} = x^\pi$.

1998 ACM Subject Classification F.4.1 Mathematical Logic; F.4.3 Formal Languages

Keywords and phrases Linear orderings, first-order logic, semigroups, rational sets

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.67

1 Introduction

One of the fundamental results in formal language theory is the equivalence between automata on finite words, rational expressions, recognizability by finite semigroups, and definability in monadic second-order logic (see e.g. [26]). This has been specified by Schützenberger [25], McNaughton and Papert [13], which show equivalence between counter-free automata, star-free expressions, recognizability by finite aperiodic semigroups, and definability in first-order logic.

These results have been extended to many classes of structures like infinite words [6, 14], bi-infinite words [10, 15], transfinite words [7, 1], traces, trees, pictures...

In [5], Bruyère and Carton introduce automata and rational expressions for words indexed by linear orderings. These notions unify naturally previously defined notions for finite words, left- and right-infinite words, bi-infinite words, and ordinal words. The question to know whether the above equivalence results hold in this setting has been addressed in several papers since then. Up to now, most results hold when one restricts to sets of words indexed by countable scattered orderings; recall that a linear ordering is scattered if it does not contain any dense sub-ordering. For this class of sets, the paper [5] already proves that a Kleene-like theorem holds. The works [23, 22] introduce a notion of $\diamond$-semigroup and show equivalence between recognizability by finite $\diamond$-semigroups and rational expressions. Finally [2] shows equivalence between rational expressions and monadic second-order logic.

Let us now consider the extension of Schützenberger-McNaughton-Papert results for sets of words indexed by countable scattered orderings. Bedon and Rispal [3] prove equivalence between star-free expressions and recognizability by finite aperiodic $\diamond$-semigroups. From

* Supported by ANR Frec

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 67–81

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

their work it is possible to obtain a characterization of star-free expressions in terms of FO definability in structures where one can quantify over elements *and cuts* of the underlying ordering. However it seems natural to consider the more classical logical framework (used e.g. in [2]) where the domain of the structure consists only of elements of the ordering, and ask for a characterization of FO-definable sets. For many classes of words (such as words indexed by ω, $\mathbb{Z}$, an ordinal, or $\mathbb{R}$ [20]), the equivalence between star-free expressions with FO logic is a relatively simple generalization of McNaughton-Papert proof. Let us try to explain why this is not the case here. A crucial point in the proof that star-free sets are FO-definable is the possibility to define in FO the product $L = L_1 \cdot L_2$ of two FO-definable sets L_1 and L_2. Intuitively, this can be done by expressing (with a FO sentence) that $w \in L$ iff there exists some position x in w such that the prefix of w which corresponds to positions before x belongs to L_1, and the remaining suffix of w belongs to L_2. The existence of x is ensured by the fact that the underlying ordering is complete. We cannot use this idea when considering any countable scattered ordering.

In this paper we characterize FO-definable sets in terms of rational expressions and recognizability by semigroups. For rational expressions, we consider the class of star-free marked sets, which is a variant of the class of star-free sets where one uses the *marked* product instead of the classical product. The operation of marked product has already been studied extensively, in particular in connection with the hierarchy of concatenation [19, Sect 7.1]. For the algebraic side, this corresponds to the class of sets which can be recognized by a $\diamond$-semigroup which is finite and aperiodic, and satisfies the additional equation $x^\omega x^{-\omega} = x^\pi$. As an immediate corollary of this characterization, we prove that it is decidable whether a rational set of words indexed by countable scattered orderings is star-free marked or, equivalently, FO-definable. We obtain as a byproduct that marked products are less expressive than usual products for linear ordering whereas they have the same expressive power for finite words.

Let us mention partial results for the class of words indexed by any linear ordering. The paper [4] introduces a new rational operation of shuffle of sets which allows to deal with dense orderings, and extends the Kleene-like theorem proved in [5] to sets of words indexed by all linear orderings. The work [2] shows that rational sets are definable in MSO logic, but not the converse (for instance, it is shown that the class of scattered orderings is MSO-definable but not rational). The decidability of FO can also be obtained with automata through linear temporal logic which is equivalent on linear orderings [9].

The paper is organized as follows. Definitions and useful results concerning linear orderings, rational sets, logic, and semigroups are recalled respectively in Sections 2, 3, 4 and 5. Section 6 states and gives a sketch of the proof of the main result.

2 Words on scattered linear orderings

2.1 Scattered linear orderings

This section recalls basic definitions on linear orderings but the reader is referred to [24] for a complete introduction. Hausdorff's characterization of countable scattered linear orderings is given and words indexed by linear orderings are introduced.

A *linear ordering* $(J, <)$ is a total ordering. A *cut* of a linear ordering J is a pair (K, L) of intervals such that $J = K \cup L$ and such that for any $k \in K$ and $l \in L$, $k < l$. The set of all cuts of the ordering J is denoted by $\hat{J}$. This set $\hat{J}$ can be linearly ordered by the relation defined by $c_1 < c_2$ if and only if $K_1 \subsetneq K_2$ for any cuts $c_1 = (K_1, L_1)$ and $c_2 = (K_2, L_2)$. This linear ordering can be extended to $J \cup \hat{J}$ by setting $j < c_1$ whenever $j \in K_1$ for any $j \in J$.

A *gap* of an ordering J is a cut (K, L) such that $K \neq \emptyset$, $L \neq \emptyset$, K has no last element and L has no first element. An ordering J is *complete* if it has no gap. For example, the linear ordering of the real numbers $\mathbb{R}$ is complete, whereas the linear ordering of the rational numbers $\mathbb{Q}$ is not.

For any linear ordering J, we denote by $-J$ the opposite linear ordering that is the set J equipped with the opposite ordering. For instance, $-\omega$ is the linear ordering of negative integers.

The sum $J + K$ of two linear orderings is the set $J \cup K$ equipped with the ordering $<$ extending the orderings of J and K by setting $j < k$ for any $j \in J$ and $k \in K$. Next, the *sum* $\sum_{j\in J} K_j$ is the set of all pairs (k, j) such that $k \in K_j$ equipped with the ordering defined by $(k_1, j_1) < (k_2, j_2)$ if and only if $j_1 < j_2$ or $(j_1 = j_2$ and $k_1 < k_2$ in $K_{j_1})$.

A linear ordering J is *dense* if for any j and k in J such that $j < k$, there exists an element i of J such that $j < i < k$. It is *scattered* if it contains no dense sub-ordering. The ordering ω of natural integers and the ordering ζ of relative integers are scattered. More generally, ordinals are scattered orderings. We denote by $\mathcal{N}$ the subclass of finite linear orderings, $\mathcal{O}$ the class of countable ordinals and $\mathcal{S}$ the class of countable scattered linear orderings. The following characterization of scattered linear orderings is due to Hausdorff.

▶ **Theorem 1.** *[Hausdorff [12]] A countable linear ordering J is scattered if and only if J belongs to $\bigcup_{\alpha\in\mathcal{O}} V_\alpha$ where the classes V_α are inductively defined by:*

1. $V_0 = \{\mathbf{0}, \mathbf{1}\}$
2. $V_\alpha = \{\sum_{j\in J} K_j \mid J \in \mathcal{N} \cup \{\omega, -\omega, \zeta\}$ *and* $K_j \in \bigcup_{\beta<\alpha} V_\beta\}$.

where $\mathbf{0}$ *and* $\mathbf{1}$ *are respectively the orderings with zero and one element.*

The *rank* $r(J)$ of a countable scattered ordering J is defined as the least ordinal α such that $J \in V_\alpha$.

2.2 Words

Let A be a finite alphabet. A *word* $w = (a_j)_{j\in J}$ indexed by a linear ordering J is a function from J to A. J is called the *length* of w. For instance ω is the length of right-infinite words $a_0a_1\cdots$ and ζ is the length of bi-infinite words $\cdots a_{-1}a_0a_1\cdots$.

The sum of linear orderings helps to define the products of words. Let J be a linear ordering and let $(x_j)_{j\in J}$ be words of respective length K_j for any $j \in J$. The word $x = \prod_{j\in J} x_j$ obtained by concatenation of the words x_j with respect to the ordering on J is of length $L = \sum_{j\in J} K_j$. For instance, if for any $j \in \omega$, we set $x_j = a^{\omega^j}$, then $x = \prod_{j\in\omega} x_j$ is the word $x = a^{\omega^\omega}$ of length $\sum_{j\in\omega} \omega^j = \omega^\omega$. The sequence $(x_j)_{j\in J}$ of words is called a *J-factorization* of the word $x = \prod_{j\in J} x_j$.

We denote by $A^\diamond$ the set of all words over A indexed by countable scattered linear orderings. The *rank* $r(w)$ of a word $w \in A^\diamond$, is, by definition the rank of its length J.

3 Rational sets of words on linear orderings

Bruyère and Carton have introduced rational expressions and automata for words indexed by countable scattered linear orderings [5]. They have proved that a set of words is described by a rational expression if and only if it is accepted by some finite automaton. Such a set is called *rational* in this paper. This result is an extension of the classical Kleene theorem on finite words. This section briefly recalls definitions of rational operations. In this paper, we

will mainly use union and concatenation but the other operations are often useful to denote sets of words.

Let A be a fixed finite alphabet. We consider the rational operations defined for any subsets X and Y of $A^\diamond$ by :

$$X + Y = \{z \mid z \in X \cup Y\} \qquad X \cdot Y = \{x \cdot y \mid x \in X, y \in Y\},$$
$$X^* = \{\textstyle\prod_{j\in\{1,\dots,n\}} x_j \mid n \in \mathcal{N}, x_j \in X\}, \qquad X^\diamond = \{\textstyle\prod_{j\in J} x_j \mid J \in \mathcal{S}, x_j \in X\},$$
$$X^\omega = \{\textstyle\prod_{j\in\omega} x_j \mid x_j \in X\}, \qquad X^{-\omega} = \{\textstyle\prod_{j\in-\omega} x_j \mid x_j \in X\},$$
$$X^\sharp = \{\textstyle\prod_{j\in\alpha} x_j \mid \alpha \in \mathcal{O}, x_j \in X\}, \qquad X^{-\sharp} = \{\textstyle\prod_{j\in-\alpha} x_j \mid \alpha \in \mathcal{O}, x_j \in X\}.$$

As usual, the dot denoting concatenation is omitted in rational expressions. A *marked product* of X and Y is a product of the form XaY for some letter $a \in A$.

The operator $\diamond$ that we have defined above is actually a special case of a more general binary operator defined in [5]. This binary operator is really needed to capture all rational sets but it is not used in this paper.

Let us define the main classes of sets that we use in this paper.

- The class $\mathrm{Rat}(A^\diamond)$ of all rational sets over A indexed by countable scattered linear orderings is the smallest set containing $\{a\}$ for any $a \in A$, the empty set, and closed under all rational operations. It is proved in [23] that this class is closed under complementation and thus under all boolean operations.
- The set $\mathrm{SF}(A^\diamond)$ of star-free sets over A indexed by countable scattered linear orderings is the smallest set containing $\{a\}$ for any $a \in A$, the empty set, and closed under product and all boolean operations.
- The set $\mathrm{SFM}(A^\diamond)$ of star-free marked sets over A indexed by countable scattered linear orderings is the smallest set containing $\{a\}$ for any $a \in A$, the empty set, and closed under *marked* product and all boolean operations.

It is interesting to note that, for finite words, the corresponding classes $\mathrm{SF}(A^*)$ and $\mathrm{SFM}(A^*)$ coincide. Any product KL is indeed equal to a finite union $K\varepsilon(L) + \sum_{a\in A} KaL_a$ where $L_a = a^{-1}L$. They do not coincide in our case as it is shown by Example 4. Let us illustrate these definitions by some examples.

▶ **Example 2.** Consider the set $X_1 \subseteq A^\diamond$ of words w over $A = \{a, b\}$ such that every position in w (apart from the last position, if any) admits a next position, and every position (apart from the first position, if any) admits a previous position. We have

$$X_1 = A^\diamond \setminus [(A^\diamond AA^\diamond)^\omega AA^\diamond + A^\diamond A(A^\diamond AA^\diamond)^{-\omega}].$$

Moreover $(A^\diamond AA^\diamond)^\omega = A^\diamond AA^\diamond \setminus A^\diamond A$ and $(A^\diamond AA^\diamond)^{-\omega} = A^\diamond AA^\diamond \setminus AA^\diamond$, thus X_1 is a star-free marked set.

▶ **Example 3.** Consider the set $X_2 = (a^\diamond aa^\diamond)^\omega (b^\diamond bb^\diamond)^{-\omega}$ of words w over $A = \{a, b\}$ which can be written as $w = w_1w_2$ where w_1 is non-empty, contains only a and has no last a and w_2 is non-empty, contains only b and has no first b. There is then a gap between w_1 and w_2. A star-free marked expression for X_2 is

$$A^\diamond aA^\diamond bA^\diamond \setminus (A^\diamond bA^\diamond aA^\diamond \cup a^\diamond ab^\diamond \cup a^\diamond bb^\diamond).$$

Observe that $a^\diamond = A^\diamond \setminus A^\diamond bA^\diamond$ and $b^\diamond = A^\diamond \setminus A^\diamond aA^\diamond$.

The following example gives a set X_3 which seems very close to the set X_2 of the previous example. This set will be star-free but our characterization of $SFM(A^\diamond)$ will allow us to prove that X_3 is not a star-free marked set (see Example 17).

▶ **Example 4.** Consider the set $X_3 = (A^\diamond AA^\diamond)^\omega (A^\diamond AA^\diamond)^{-\omega}$ of words w over $A = \{a, b\}$ such that the underlying ordering of w contains at least one gap. We have $X_3 \in SF(A^\diamond)$.

4 Logic

Let us recall useful elements of logic, and settle some notations. For more details we refer e.g. to Thomas' survey paper [26].

We consider first-order (shortly: FO) logic over relational signatures. As usual, we will often identify logical symbols with their interpretation. We call FO sentence every FO formula without free variable.

For every finite alphabet $A = \{a_1, \ldots, a_n\}$ we consider the relational signature $\mathcal{L}_A = \{<, P_{a_1}, \ldots, P_{a_n}\}$ where $<$ denotes a binary predicate symbol and every P_{a_i} denotes a unary predicate symbol. One can associate to every word $w = (a_j)_{j\in J}$ over A (where $a_j \in A$ for every j) the $\mathcal{L}_A$-structure $M_w = (J; <, (P_a)_{a\in A})$ where $<$ is interpreted as the ordering over J, and $P_a(x)$ holds if and only if $a_x = a$. In order to take into account the case $w = \varepsilon$, which leads to the structure M_ε which has an empty domain, we will allow structures to be empty.

Given an FO sentence φ over the signature $\mathcal{L}_A$, we define the set L_φ as the set of words $w \in A^\diamond$ such that $M_w \models \varphi$. This definition extends to the case of FO formulas with free variables. For every word $w = (a_j)_{j\in J}$ over A and every n-tuple $b_1, \ldots, b_n$ of elements of J, we define $w(b_1, \ldots, b_n)$ as the word $w' = (a'_j)_{j\in J}$ over the alphabet $\{0, 1\}^n \times A$ such that for every $j \in J$, the last component of a'_j equals a_j, and for every $i \in \{1, \ldots, n\}$, the i-th component of a'_j equals 1 if and only if $j = b_i$. Now, given a FO formula $\varphi(x_1, \ldots, x_n)$ with free variables $x_1, \ldots, x_n$, we define L_φ as the class of words of the form $w(b_1, \ldots, b_n)$ over the alphabet $\{0, 1\}^n \times A$ such that $M_w \models \varphi(b_1, \ldots, b_n)$.

We will say that a set $X \subseteq A^\diamond$ is *FO-definable* if there exists an FO-formula φ over the signature $\mathcal{L}_A$ such that $X = L_\varphi$.

▶ **Example 5.** The set X_1 of Example 2 is FO definable. We first define the auxiliary predicate $suc(x, y)$ as $x < y \wedge \neg\exists z(x < z \wedge z < y)$. Then X_1 is definable by the formula

$$\forall x[(\exists y\ x < y \longrightarrow \exists y\ suc(x, y)) \wedge (\exists y\ y < x \longrightarrow \exists y\ suc(y, x))].$$

▶ **Example 6.** The set X_2 of Example 3 can be defined by the FO-formula

$$\exists x\ P_a(x) \wedge \exists y\ P_b(y) \wedge \neg\exists x\exists y(x < y \wedge P_b(x) \wedge P_a(y)) \quad (1)$$
$$\wedge\forall x(P_a(x) \to \exists y(y > x \wedge P_a(y))) \quad (2)$$
$$\wedge\forall y(P_b(y) \to \exists x(x < y \wedge P_b(x))) \quad (3)$$

The sub-formula (1) expresses that the word contains some a and some b, and that no a occurs after some b. The sub-formula (2) (resp. (3)) ensures that there is no last a (resp. no first b).

5 Algebraic characterization of rational sets

The algebraic objets that we use to characterize FO over countable scattered linear orderings are semigroups enriched with operations that make them suitable for linear orderings. We start with the definition of a semigroup.

A semigroup is a set S equipped with an associative binary product. Since the product is associative, the product $s_1 \cdots s_n$ of n elements $s_1, \ldots, s_n$ is well-defined. The semigroup S^1 is S if S has already a neutral element and it is the semigroup obtained by adding a neutral element otherwise. An *idempotent* e of a semigroup is an element such that $e^2 = e$.

5.1 $\diamond$-semigroups

The product of semigroups is generalized to recognize sets of words indexed by countable scattered linear orderings. A $\diamond$-semigroup is a generalization of a usual semigroup. The product of a sequence indexed by any scattered ordering is defined. For any set S, recall that $S^\diamond$ denotes the set of words over S indexed by any countable scattered linear ordering.

▶ **Definition 7.** A *$\diamond$-semigroup* is a set S equipped with product $\pi : S^\diamond \to S$ which maps any word of $S^\diamond$ to an element of S such that

- for any element s of S, $\pi(s) = s$;
- for any word x of $S^\diamond$ and for any factorization $x = \prod_{j \in J} x_j$ where $J \in S$,

$$\pi(x) = \pi(\prod_{j \in J} \pi(x_j)).$$

The latter condition is a generalization of associativity. It states that for any factorization $x = \prod_{j \in J} x_j$ of some word $x \in S^\diamond$, the product of x can be obtained by first computing the product $\pi(x_j)$ of each word x_j to get a word $y = \prod_{j \in J} \pi(x_j)$ of length J and then computing the product $\pi(y)$ of that word y.

Note that a $\diamond$-semigroup (S, π) is already a semigroup. For any two elements s and t of S, the finite product $\pi(st)$ (more precisely, the product of the sequence st of length 2) is defined. It is merely denoted by st. The generalization of associativity ensures that $r(st) = \pi(r\pi(st)) = \pi(rst) = \pi(\pi(rs)t) = (rs)t$ for any $r, s, t \in S$.

The set $A^\diamond$ equipped with the concatenation is a $\diamond$-semigroup. All $\diamond$-semigroups considered in this paper are either of the form $A^\diamond$ for some alphabet A or they are finite. The following example is a $\diamond$-semigroup where the underlying set S is finite (these $\diamond$-semigroups will be studied in Section 5.2).

▶ **Example 8.** The set $S = \{0, 1\}$ equipped with the product π defined for any $u \in S^\diamond$ by $\pi(u) = 0$ if u has at least one occurrence of the letter 0, and $\pi(u) = 1$ otherwise, is a $\diamond$-semigroup.

A *sub-$\diamond$-semigroup* T of a $\diamond$-semigroup S is a subset of S closed under product. A *morphism of $\diamond$-semigroup* is an application which preserves the product. A function $\varphi : S \to T$ is a morphism from (S, π_S) to (T, π_T) if for any word $x = (s_j)_{j \in J}$, one has $\pi_T(\varphi(x)) = \varphi(\pi_S(x))$ where $\varphi(x) = (\varphi(s_j))_{j \in J}$. A $\diamond$-semigroup T is a *quotient* of a $\diamond$-semigroup S if there exists an onto morphism from S to T. A $\diamond$-semigroup T *divides* S if T is the quotient of a sub-$\diamond$-semigroup of S.

5.2 Finite $\diamond$-semigroups

A $\diamond$-semigroup (S, π), of course, is said to be finite if S is finite. Even when S is finite, the function π is not easy to describe because the product of any sequence has to be given. It turns out that the function π can be described using a semigroup structure on S with two additional functions from S to S. This gives a finite description of the product π.

It has already been noted that a $\diamond$-semigroup (S, π) has already a structure of semigroup since π is defined on words of length 2. Let us define two functions $\tau : S \to S$ and $-\tau : S \to S$.

The images of these functions are denoted using exponentiation : $\tau : s \mapsto s^\tau$ and $-\tau : s \mapsto s^{-\tau}$. For any $s \in S$, s^τ and $s^{-\tau}$ are respectively equal to $\pi(s^\omega)$ and $\pi(s^{-\omega})$ where $s^\omega = sss\cdots$ and $s^{-\omega} = \cdots sss$ are the two words of length ω and $-\omega$ in which s occurs at all positions.

The functions τ and $-\tau$ satisfy the following equations. For any $s, t \in S$ and for any integer n, one has $s(ts)^\tau = (st)^\tau$, $(s^n)^\tau = s^\tau$, $(st)^{-\tau}s = (ts)^{-\tau}$ and $(s^n)^{-\tau} = s^{-\tau}$. Equations for τ follow from the equality between the ω-words $(s^n)^\omega$ and s^ω and from the equality between the ω-words $(st)^\omega$ and $s(ts)^\omega$. Equations for $-\tau$ follow from similar relations for words of lengths $-\omega$. Functions satisfying these equations are respectively called *compatible to the right* and *compatible to the left* with S.

Note that these two functions τ and $-\tau$ can be defined even when the $\diamond$-semigroup is not finite. When the $\diamond$-semigroup (S, π) is finite, the semigroup structure of S and the two functions τ and $-\tau$ completely describe its product π. This is stated in the following theorem.

▶ **Theorem 9** ([23, 22]). *Let S be a finite semigroup and let τ and $-\tau$ be functions respectively compatible to the right and to the left with S. Then S can be uniquely endowed with a structure of $\diamond$-semigroup (S, π) such that $s^\tau = \pi(s^\omega)$ and $s^{-\tau} = \pi(s^{-\omega})$.*

The previous theorem means that a finite $\diamond$-semigroup has a finite description. It suffices to give a semigroup product and two compatible functions to fully characterize the product.

We briefly explain how the product π can be recovered from the semigroup structure and the compatible functions. The construction of π is based on the next Lemma which follows directly from Ramsey's Theorem [21].

Let S be a semigroup. We denote by φ the natural morphism from S^* to S which maps any finite sequence of elements to their product. Let $x = (s_j)_{j\in\omega}$ be an ω-word over S. A ω-factorization of x is ω-sequence $(x_j)_{j\in\omega}$ of finite words such that $x = \prod_{j\in\omega} x_j$. A *right linked pair* of a semigroup S is a pair (s, e) such that $se = s$ and $e^2 = e$.

▶ **Lemma 10.** *For any ω-word x over a semigroup S, there is an ω-factorization $x = \prod_{j\in\omega} x_j$ and a right linked pair (s, e) such that $\varphi(x_0) = s$ and $\varphi(x_j) = e$ for any $j \geq 1$.*

Such a factorization is called a *ramseyan factorization*, see Theorem 3.2 in [17]. If $x = \prod_{j\in\omega} x_j$ is a ramseyan factorization of x, then $\pi(x)$ must be equal to se^τ since π satisfies a generalized associativity. The product π is then defined on all words in $S^\diamond$ by induction on their rank. A word of rank α is, indeed, either a finite, or an ω , or a $-\omega$-product of words of ranks smaller than α.

Note that a given ω-word over S may have several ramseyan factorizations related to different right linked pairs (s_1, e_1) and (s_2, e_2). It turns out that these linked pairs are then conjugated. There exist elements $x, y \in S^1$ such that $s_1 x = s_2$, $e_1 = xy$ and $e_2 = yx$. Since the function τ is compatible, one has $s_1 e_1^\tau = s_1(xy)^\tau = s_1 x(yx)^\tau = s_2 e_2^\tau$.

The functions τ and $-\tau$ are usually denoted ω and $-\omega$. This may cause a small confusion since s^ω is either an ω-word over S or its product in S but it is always clear from the context which one is meant.

▶ **Example 11.** Consider again the $\diamond$-semigroup $S = \{0, 1\}$ of Example 8. Its semigroup structure is $\{0, 1\}$ with the usual multiplication ($11 = 1$ and $00 = 01 = 10 = 0$). The compatible functions ω and $-\omega$ are defined by $0^\omega = 0^{-\omega} = 0$ and $1^\omega = 1^{-\omega} = 1$.

▶ **Example 12.** The set $S = \{0, 1\}$ equipped with the product π defined for any $u \in S^\diamond$ by $\pi(u) = 1$ if only 1 occurs in u and if the length of u is an ordinal and $\pi(u) = 0$ otherwise, is a $\diamond$-semigroup. Its semigroup structure is again $\{0, 1\}$ with the usual multiplication but the compatible functions ω and $-\omega$ defined by $0^\omega = 0^{-\omega} = 1^{-\omega} = 0$ and $1^\omega = 1$.

Since any element s of a finite semigroup has a power s^n which is an idempotent and since $(s^n)^\omega = s^\omega$, it suffices to give the values of e^ω when e is an idempotent to completely describe the function ω. The same applies to the function $-\omega$.

5.3 Recognizability

It is well known that rational sets of finite words are exactly those recognized by finite semigroups (see e.g. [18]). This result can be generalized for words indexed by countable scattered linear orderings.

Let S and T be two $\diamond$-semigroups. The $\diamond$-semigroup T *recognizes* a subset X of S if and only if there exists a morphism $\varphi : S \to T$ and a subset $P \subseteq T$ such that $X = \varphi^{-1}(P)$.

The following theorem states that finite $\diamond$-semigroups are equivalent to rational expressions and automata for words indexed by countable scattered orderings.

▶ **Theorem 13** ([23]). *A set $X \subseteq A^\diamond$ is rational if and only if it is recognized by a finite $\diamond$-semigroup.*

It is also proved in [23] that any rational set X of words over countable scattered linear orderings has a *syntactic $\diamond$-semigroup*. This is a smallest $\diamond$-semigroup recognizing X in a strong way. Not only it is the smallest in cardinality but it also divides any other $\diamond$-semigroup recognizing X. As in the case of finite words, this syntactic $\diamond$-semigroup can be obtained by quotienting any $\diamond$-semigroup recognizing X by the relation that identifies elements which cannot be distinguished by contexts (intuitively, contexts are terms which involve ω and $-\omega-$products, and with a hole in it).

The following example shows how the $\diamond$-semigroup introduced in Example 12 can be used to recognize the set of words of ordinal length.

▶ **Example 14.** Consider the $\diamond$-semigroup S defined in Example 12 and the morphism $\varphi : A^\diamond \to S$ defined by $\varphi(a) = 1$ for any $a \in A$. The set of words with countable ordinal length is recognizable since it is equal to $\varphi^{-1}(\{1\})$. It will be shown after Theorem 21 that this set is not FO-definable. This is a variant of Tarski's result (see [24, Theorem 13.13]) that the class of well-orderings is not elementary.

We give below some examples of morphisms from $A^\diamond$ into finite $\diamond$-semigroups that recognize subsets of $A^\diamond$ that have been already encountered in Examples 2, 3 and 4. It can be checked that, in each example, the given $\diamond$-semigroup is actually the syntactic $\diamond$-semigroup of the set it recognizes. For each $\diamond$-semigroup, we give the $\mathcal{D}$-classes structure.

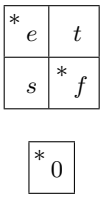


Figure 1 $\mathcal{D}$-classes structure of $\diamond$-semigroup of Example 15

▶ **Example 15.** The set X_1 of Example 2 is recognized by the $\diamond$-semigroup $S_1 = \{0, e, t, s, f\}$ whose product is defined by $ts = e^2 = e$, $et = tf = t$, $se = fs = s$, $st = f^2 = f$, $e^\omega = t$, $e^{-\omega} = s$, $f^\omega = f^{-\omega} = f$ and any other product is equal to 0. Define the morphism

$\varphi_1 : A^\diamond \to S_1$ by $\varphi_1(a) = \varphi_1(b) = e$. Observe that $\varphi_1^{-1}(e)$ is the set of words which have a first and a last element, and $\varphi_1^{-1}(f)$ is the set of words which have neither a first nor a last element. We have $X_1 = \varphi_1^{-1}(S_1 \setminus \{0\})$.

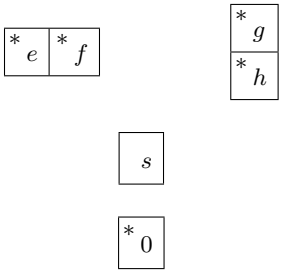


Figure 2 $\mathcal{D}$-classes structure of $\diamond$-semigroup of Example 16

▶ **Example 16.** The set X_2 of Example 3 is recognized by the $\diamond$-semigroup $S_2 = \{0, e, f, g, h, s\}$ whose product is defined by $e^2 = fe = e^{-\omega} = e$, $f^2 = ef = e^\omega = f^\omega = f^{-\omega} = f$, $g^2 = gh = g^\omega = g$, $h^2 = hg = h^\omega = g^{-\omega} = h^{-\omega} = h$, $fh = es = fs = sg = sh = s$, and any other product is equal to 0. Define the morphism $\varphi_2 : A^\diamond \to S_2$ by $\varphi_2(a) = e$ and $\varphi_2(b) = g$. We have $X_2 = \varphi_2^{-1}(s)$. It will be seen after Theorem 21 that X_2 is FO-definable.

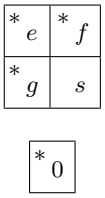


Figure 3 $\mathcal{D}$-classes structure of $\diamond$-semigroup of Example 17

▶ **Example 17.** The set X_3 of Example 4 is recognized by the $\diamond$-semigroup $S_3 = \{0, e, f, g, s\}$ whose product is defined by $e^2 = fe = eg = e$, $f^2 = ef = es = e^\omega = f^\omega = f$, $g^2 = ge = se = e^{-\omega} = g^{-\omega} = g$, $gs = sf = gf = g^\omega = f^{-\omega} = s$ and any other product is equal to 0. Define the morphism $\varphi_3 : A^\diamond \to S_3$ by $\varphi_3(a) = \varphi_3(b) = e$. We have $X_3 = \varphi_3^{-1}(0)$. Our characterization of FO-definability (Theorem 21) will imply that X_3 is not FO-definable.

5.4 Equivalence between rational sets, $\diamond$-semigroups, and logic

The equivalence between rational expressions, $\diamond$-semigroups and logic was proved in [23, 22, 2]. The logical side involves Monadic Second-Order (shortly: MSO) logic. Recall that MSO logic is an extension of first-order logic that allows to quantify over elements as well as subsets of the domain of the structure. The notion of MSO-definable set extends in a natural way the one of FO-definable set. For more details about MSO logic we refer e.g. to [11, 26].

The following theorem is an extension of the classical theorem of Büchi [6] which states that, for finite words, MSO exactly defines rational sets of words.

▶ **Theorem 18** ([23, 22, 2]). *Let A be a finite alphabet, and let $X \subseteq A^\diamond$ be a set of words indexed by countable scattered linear orderings. Then the following properties are equivalent:*

1. *X is rational;*
2. *X is recognizable by a finite $\diamond$-semigroup;*
3. *X is MSO-definable.*

Bedon and Rispal [3] extended Schützenberger' theorem [25] to the class of sets of words indexed by countable scattered linear orderings. In order to state their result, we recall the definitions of an aperiodic semigroup and of an aperiodic $\diamond$-semigroup.

▶ Proposition 19. ([17, Annex A,Prop. 2.9]) Let S be a finite semigroup. The following conditions are equivalent.

1. There exists an integer n such that $s^{n+1} = s^n$ for any $s \in S$;
2. every group in S is trivial;
3. Each $\mathcal{H}$-class is trivial.

A semigroup S satisfying these conditions is called *aperiodic*.

Note that if $s^{n+1} = s^n$, then $s^{m+1} = s^m$ for any $m \geq n$. If a finite semigroup S is aperiodic, then $s^{n+1} = s^n$ for any $s \in S$ and for any large enough integer n. Such an integer is traditionally denoted by ω in semigroup theory but we will denote it π. This symbol π is also used for the product of the $\diamond$-semigroup but this will not lead to ambiguous interpretations in the sequel. A $\diamond$-semigroup S is said to be aperiodic whenever its semigroup structure is an aperiodic semigroup. This definition allows us to state the characterization of star-free sets due to Bedon and Rispal.

▶ **Theorem 20** ([3]). *Let A be a finite alphabet, and let $X \subseteq A^\diamond$ be a set of words indexed by countable scattered linear orderings. Then the following properties are equivalent:*

1. *X is star-free;*
2. *X is recognizable by a $\diamond$-semigroup which is finite and aperiodic;*
3. *X can be defined by FO over the cuts.*

In the previous theorem, "defined by FO over the cuts" means that X is defined by a first order formula with quantification over positions of the words, (that is, elements of its length) but also over cuts of its length. This is not, strictly speaking, FO since the cuts are not part of the structure $\mathcal{M}_w$ of a word w. The last statement is not given in [3] but the equivalence between star-freeness and FO over the cuts is not difficult. In this paper, we give the equivalence between marked star-freeness and FO (without the cuts). The proof carries easily over star-freeness and the cuts.

For sets of finite words, McNaughton-Papert theorem [13] states that star-free sets coincide with FO-definable sets. For sets of words indexed by countable scattered linear orderings, one can prove that FO-definable sets are star-free (see Proposition 23), but the converse does not hold anymore. For instance, it is easy to check that the set $X = \{a^\omega\}$ is star-free, but it can be shown that X is not FO-definable (this comes from the fact that the ordering ω is undistinguishable from any ordering of the form $\omega + \zeta \times \alpha$ in FO logic, see e.g. [24, Proposition 6.12]). In the next section we provide a characterization of FO-definable sets.

6 Main result

We finally come to the main result of the paper, characterization of FO for words over countable scattered linear orderings

▶ **Theorem 21.** *Let A be a finite alphabet, and let $X \subseteq A^\diamond$ be a set of words indexed by countable scattered linear orderings. Then the following properties are equivalent:*

1. *X is a star-free marked set;*

2. *X is FO-definable;*
3. *X is recognizable by a finite aperiodic $\diamond$-semigroup satisfying the equation $x^\omega x^{-\omega} = x^\pi$;*
4. *the syntactic $\diamond$-semigroup of X is finite, aperiodic and satisfies the equation $x^\omega x^{-\omega} = x^\pi$.*

In the sequel, a finite aperiodic $\diamond$-semigroup satisfying the equation $x^\omega x^{-\omega} = x^\pi$ is called an *FO-semigroup*. The theorem is illustrated by the following examples.

The set X_1 of Example 2 is star-free marked, FO-definable (see Example 5) and the $\diamond$-semigroup provided in Example 15 is a FO-semigroup. Similarly, the set X_2 of Example 3 is star-free marked, FO-definable (see Example 6) and the $\diamond$-semigroup provided in Example 16 is a FO-semigroup. On the other hand, the set X_3 of Example 4 is star-free but its syntactic $\diamond$-semigroup given in Example 17 is not a FO-semigroup. Indeed we have $e^\omega e^{-\omega} = fg = 0 \neq e^\pi$ since $e^\pi = e$. Thus X_3 is not a star-free marked set, and is not FO-definable.

Theorem 21 yields an effective procedure to test whether a rational set $X \subseteq A^\diamond$ is star-free marked. Indeed Theorem 18 allows to compute effectively the finite syntactic $\diamond-$semigroup of X, from which one can decide whether S is aperiodic and satisfies the equation $x^\omega x^{-\omega} = x^\pi$.

▶ **Corollary 22.** *Let $X \subseteq A^\diamond$ be a rational set of words indexed by countable scattered linear orderings. Then it is decidable whether X is FO-definable.*

The proof of Theorem 21 is organized as follows. Section 6.1 proves the equivalence between (1) and (2). This is a straightforward generalization of the case of sets of finite words. In Proposition 24 we prove that (1) implies (3); it is again a rather easy extension of the case of finite words. The most difficult part is to prove that (3) implies (1), namely that sets recognizable by FO-semigroups are star-free marked sets. The proof is long and technically involved. It relies on the study of the structure of $\mathcal{D}$-classes of a FO-semigroup which recognizes the set. In Section 6.2.2 we give the general structure of the proof, but details are omitted.

6.1 First-order logic vs star-free marked sets

In this section we state equivalence between star-free marked expressions and FO-definability.

▶ Proposition 23. Let A be a finite alphabet, and let $X \subseteq A^\diamond$ be a set of words indexed by countable scattered linear orderings. The set X is a star-free marked set if and only if X is definable in first-order logic.

Proof. (sketch) The proof is a straightforward generalization of the proof of McNaughton-Papert Theorem given in [26, Theorem 4.4].

The "only if" part is proved by induction on a star-free marked expression denoting X.

For the converse, consider a first-order formula $\varphi(x_1, \ldots, x_n)$ with quantifier-depth m, and assume (without loss of generality) that φ holds only if $x_1 < x_2 \cdots < x_n$. Then one can prove by induction on m (using Ehrenfeucht-Fraïssé games) that φ is equivalent to a disjunction of formulas of the form

$$\psi_0 \wedge P_{a_1}(x_1) \wedge \psi_1 \wedge \cdots \wedge P_{a_n}(x_n) \wedge \psi_n$$

where all formulas ψ_i have quantifier depth m, and for every $1 \leq i \leq n-1$ (respectively $i = 0$, $i = n$), ψ_i is a formula where all quantifiers are relativized to the interval (x_i, x_{i+1}), except for ψ_0 (respectively ψ_n) for which all quantifiers are relativized to elements less than x_0 (respectively greater than x_n).

Now assume that X is FO-definable by a sentence ψ. Assume first that ψ is of the form $\exists x \varphi(x)$. Then using the above result, ψ is equivalent to a disjunction of formulas of the form

$\exists x(\psi_0 \wedge P_{a_i}(x) \wedge \psi_1)$. Each such formula defines a star-free marked set, thus $X \in SFM(A^\diamond)$. In case ψ has the form $\forall x\varphi(x)$, we use the equivalence $\psi \equiv \neg\exists x\neg\varphi(x)$. ◀

6.2 FO-semigroups vs star-free marked sets

6.2.1 From star-free marked sets to FO-semigroups

▶ Proposition 24. Let A be a finite alphabet, and let $X \subseteq A^\diamond$ be a set of words indexed by countable scattered linear orderings. If X is a star-free marked set then X is recognizable by a FO-semigroup.

The proof is very close to the one given in [16] for the case of sets of finite words. It goes by induction on a star-free marked expression denoting the set X. Here we have to show, in addition, that a $\diamond$-semigroup S which recognizes X satisfies the equation $x^\omega x^{-\omega} = x^\pi$.

For every set $X \subseteq A^\diamond$ recognizable by a finite aperiodic $\diamond$-semigroup S, we define $i(X)$ as the least integer n such that for every $x \in A^\diamond$ and every context C, we have $C(x^{n+1}) \in X \Leftrightarrow C(x^n) \in X$ (let us recall that a context is, intuitively, a term with a hole in it).

Proof. The proof goes by induction on a star-free marked expression denoting X.

The cases when $X = \emptyset$, and $X = \{a\}$ with $a \in A$, are easy.

Assume now that X_1 and X_2 are star-free marked sets which are recognized by the FO-semigroup S_1 and S_2, respectively. The proof that the sets $Y_1 = X_1 + X_2$ and $Y_2 = A^\diamond \setminus X_1$ are recognizable by a FO-semigroup is easy. They are both recognized by the $\diamond$-semigroup $S_1 \times S_2$ with the component-wise product. This $\diamond$-semigroup is obviously a FO-semigroup.

Let us prove that every set $X \subseteq A^\diamond$ of the form $X = X_1 a X_2$ with $a \in A$, is recognizable by a FO-semigroup.

Let S be a finite $\diamond$-semigroup which recognizes X, and let $\varphi : A^\diamond \to S$ be the associated morphism. Let us show that S is aperiodic with $i(X) \leq i(X_1) + i(X_2) + 1$. This amounts to show that for all words $u, v, w \in A^\diamond$ and every integer $n \geq i(X_1) + i(X_2) + 1$, one has $uv^n w \in X$ if and only if $uv^{n+1}w \in X$. It is actually sufficient to prove that if $uv^n w \in X$, then $uv^{n+1}w \in X$ since the finiteness of S implies that there always exists an integer $p \geq 1$ such that for n large enough, one has $uv^n w \in X$ if and only if $uv^{n+p}w \in X$. Assume first that $uv^n w \in X$. By definition of X there exists $z_1 \in X_1$ and $z_2 \in X_2$ such that $uv^n w = z_1 a z_2$. We consider several cases:

- if $uv^n w' = z_1$ for some prefix w' of w, then it follows from our hypothesis on X_1 and the fact that $n \geq i(X_1)$ that $uv^{n+1}w' \in X_1$, thus $uv^{n+1}w \in X$.
- The case when $u'v^n w = z_2$ for some suffix u' of u is similar to the previous case.
- Assume now that $z_1 = uv^{n_1}v_1$ and $z_2 = v_2 v^{n_2} w$ with $v_1 a v_2 = v$ and $n_1 + n_2 + 1 = n$. By hypothesis we have $n_1 + n_2 + 1 \geq i(X_1) + i(X_2) + 1$, thus either $n_1 \geq i(X_1)$, or $n_2 \geq i(X_2)$. If $n_1 \geq i(X_1)$ then it follows from our hypothesis on X_1 that $uv^{n_1+1}v_1 \in X_1$, which yields $uv^{n+1}w \in X$. The case when $n_2 \geq i(X_2)$ is similar.

Let us now prove that $x^\omega x^{-\omega} = x^\pi$ for every $x \in S$. This amounts to show that for all words $u, v, w \in A^\diamond$, there exists an integer $n > i(X)$ such that $uv^\omega v^{-\omega} w \in X$ if and only if $uv^n w \in X$. The case $v = \varepsilon$ is trivial. We suppose now that $v \neq \varepsilon$.

Assume first that $uv^\omega v^{-\omega} w \in X$. By definition of X there exists $z_1 \in X_1$ and $z_2 \in X_2$ such that $uv^\omega v^{-\omega} w = z_1 a z_2$. We consider several cases:

- if uv^ω is a prefix of z_1, then it is a strict prefix since az_2 cannot be equal to $v^{-\omega}w$. Thus $z_1 = uv^\omega v^{-\omega} w'$ for some prefix w' of w. It follows from our hypothesis on X_1 that for every $n \geq i(X_1)$ we have $uv^n w' \in X_1$, thus $uv^n w \in X$.

- the case when z_1 is a prefix of uv^ω is similar.

Note that we have really used here that this is a marked product.

Conversely assume that $uv^n w \in X$ for some integer $n > i(X)$, and let $uv^n w = z_1 a z_2$ with $z_1 \in X_1$ and $z_2 \in X_2$. By definition of $i(X)$ we can assume that $n \geq i(X_1) + i(X_2) + 1$. Let $y = uv^n w$.

If y can be written as $y = uv^{n_1} w_1 w_2$ with $n_1 \geq i(X_1)$ and $uv^{n_1} w_1 \in X_1$, then the induction hypothesis implies that $uv^\omega v^{-\omega} w_1 \in X_1$, hence $uv^\omega v^{-\omega} w \in X$.

The case where y can be written as $y = u_1 u_2 v^{n_2} w$ with $n_2 \geq i(X_2)$ and $u_2 v^{n_2} w \in X_2$ is similar. ◀

6.2.2 From FO-semigroups to star-free marked sets

In this section we state the following result, and discuss about its proof.

▶ Proposition 25. Let A be a finite alphabet, and let $X \subseteq A^\diamond$ be a set of words indexed by countable scattered linear orderings. If X is recognizable by a FO-semigroup then X is a star-free marked set.

Let us explain the main ingredients of the proof of Proposition 25. The structure of the proof is similar to the one of the proof of Schützenberger's theorem given in [16]. The proof goes by induction on the $\mathcal{D}$-classes of the FO-semigroup recognizing X.

Let us give some details. Assume that X is recognized by the morphism $\varphi : A^\diamond \to S$ into a FO-semigroup S. There exists a subset P of S such that $X = \varphi^{-1}(P)$. Since $X = \bigcup_{s \in P} \varphi^{-1}(s)$ and since star-free marked sets are closed under union, it is sufficient to prove that $\varphi^{-1}(s)$ is a star-free marked set for each $s \in S$.

For every subset P of S, let $X_P = \varphi^{-1}(P)$. In case P is reduced to a singleton set $\{s\}$, we simply denote X_P as X_s. We shall prove that for every $s \in S$, the set X_s is marked star-free. This is proved by induction on the integer $h(s) = |S| - |S^1 s S^1|$ where $|P|$ denotes the cardinality of P.

The following definition is frequently used in the proof. Let X and Y be two subsets of $A^\diamond$ and let D be a $\mathcal{D}$-class of S. We say that Y is a *D-approximation of X* if

$$X \subseteq Y \subseteq X \cup \varphi^{-1}(\{s \mid s <_{\mathcal{J}} D\})$$

We often use this definition when $X = \varphi^{-1}(P)$ for some subset P of D.

The main difference with the proof of Schützenberger's theorem in [16], is that sets of the form $X_s X_t$ for $s, t \in S$ appear in some rational expressions. Since these products are not marked, it is necessary to prove that these sets are also star-free marked. To cope with this problem, we actually prove by induction on $k \geq 0$ the following two statements.

(P_1) for every $s \in S$, if $h(s) \leq k$ then X_s is a star-free marked set.

(P_2) for all $s, t \in S$, if $h(s) \leq k$, $h(t) \leq k$ and $h(st) > k$ then $X_s X_t$ is a star-free marked set.

Note that $X_s X_t$ is contained in X_{st}. If $h(st) \leq k$, the set $X_s X_t$ can always be replaced in expressions by X_{st}, which is already star-free marked by (P_1).

We do not discuss the case $k = 0$ which is easy. Assume now that $k > 0$. Observe first that the set $\{s \mid h(s) = k\}$ is a union of $\mathcal{D}$-classes. Indeed for all $s, t \in S$, the relation $s \mathcal{D} t$ implies $h(s) = h(t)$. Let D be one of these $\mathcal{D}$-classes and let s_0 be an element of D. Let R and L be the $\mathcal{R}$-class and the $\mathcal{L}$-class of s_0, respectively. The main steps of the proof of (P_1) are the following.

1. We show that there exist two star-free marked sets Y_R and Y_L which are D-approximations of X_R and X_L. Since S is aperiodic, we have $\{s_0\} = R \cap L$, and $Y_R \cap Y_L$ is a D-approximation of X_{s_0}.

2. We show that the set $Z = \varphi^{-1}(\{s \mid s <_{\mathcal{J}} D\})$ is star-free marked set. This gives the equality $X_{s_0} = (Y_R \cap Y_L) \setminus Z$ which shows that X_{s_0} is star-free marked.

7 Conclusion

Let us mention a few problems that are raised by our work.

We proved that for countable scattered orderings, FO logic captures the class of star-free marked sets. Which extension of FO does capture the class of star-free sets ? By [2], we know that this logic is a strict fragment of MSO. It can be shown that the existential fragment of MSO is not convenient, since for instance even the set $\{a^\omega\}$ is not definable in this fragment.

In the case of finite words, some subclasses of FO have already been algebraically characterized. Let us mention, for instance, that FO with two variables, usually called FO^2, correspond to a class of semigroups called DA. It would be interesting to know whether this is still true for linear orderings.

A lot of results concerning FO over linear orderings are obtained with Ehrenfeucht-Fraïssé games [24]. Some of them may deserve to be reconsidered using an algebraic approach.

Another interesting question is to remove the hypothesis *countable* or *scattered*. Very recently, the second author, together with Colcombet and Puppis [8], have extended the algebraic framework, and also the equivalence with MSO logic, to the case of all countable orderings. Does the characterization of FO still hold in that framework ?

Acknowledgements We thank the anonymous referees for useful suggestions.

References

1 N. Bedon. Finite automata and ordinals. *Theoret. Comput. Sci.*, 156:119–144, 1996.

2 N. Bedon, A. Bès, O. Carton, and C. Rispal. Logic and rational languages of words indexed by linear orderings. *Theory of Computing Systems*, 46(4):737–760, 2010.

3 N. Bedon and C. Rispal. Schützenberger and Eilenberg theorems for words on linear orderings. In C. De Felice and A. Restivo, editors, *DLT'2005*, volume 3572 of *Lect. Notes in Comput. Sci.*, pages 134–145. Springer-Verlag, 2005.

4 A. Bès and O. Carton. A Kleene theorem for languages of words indexed by linear orderings. *Int. J. Found. Comput. Sci.*, 17(3):519–542, 2006.

5 V. Bruyère and O. Carton. Automata on linear orderings. *J. Comput. System Sci.*, 73(1):1–24, 2007.

6 J. R. Büchi. On a decision method in the restricted second-order arithmetic. In *Proc. Int. Congress Logic, Methodology and Philosophy of science, Berkeley 1960*, pages 1–11. Stanford University Press, 1962.

7 J. R. Büchi. Transfinite automata recursions and weak second order theory of ordinals. In *Proc. Int. Congress Logic, Methodology, and Philosophy of Science, Jerusalem 1964*, pages 2–23. North Holland, 1965.

8 O. Carton, T. Colcombet, and G. Puppis. Regular languages of words over countable linear orderings. 2011. submitted.

9 J. Cristau. Automata and temporal logic over arbitrary linear time. *CoRR*, abs/1101.1731, 2011.

10 D. Girault-Beauquier. Bilimites de langages reconnaissables. *Theoret. Comput. Sci.*, 33(2–3):335–342, 1984.

11 Y. Gurevich. Monadic second-order theories. In J. Barwise and S. Feferman, editors, *Model-Theoretic Logics*, pages 479–506. Springer-Verlag, Perspectives in Mathematical Logic, 1985.

12 F. Hausdorff. *Set Theory*. Chelsea, New York, 1957.

13 R. McNaughton and S. Papert. *Counter free automata*. MIT Press, Cambridge, MA, 1971.

14 D. E. Muller. Infinite sequences and finite machines. In *Switching Circuit Theory and Logical Design: Proc. Fourth Annual Symp.*, pages 3–16. I.E.E.E., New York, 1963.

15 M. Nivat and D. Perrin. Ensembles reconnaissables de mots bi-infinis. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 47–59, 1982.

16 D. Perrin. Finite automata. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 1, pages 1–57. Elsevier, 1990.

17 D. Perrin and J.-É. Pin. *Infinite Words*. Elsevier, 2004.

18 J.-É. Pin. *Variétés de Langages Formels*. Masson, Paris, 1984.

19 J.-É. Pin. Syntactic semigroups. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 679–746. Springer-Verlag, 1997.

20 A. Rabinovich. Star free expressions over the reals. *Theoret. Comput. Sci.*, 233(1–2):233–245, 2000.

21 F. P. Ramsey. On a problem on formal logic. *Proc. London Math. Soc.*, 30(2):264–286, 1929.

22 C. Rispal. *Automates sur les ordres linéaires: complémentation*. PhD thesis, University of Marne-la-Vallée, France, 2004.

23 C. Rispal and O. Carton. Complementation of rational sets on countable scattered linear orderings. In C. S. Calude, E. Calude, and M. J. Dinneen, editors, *DLT'2004*, volume 3340 of *Lect. Notes in Comput. Sci.*, pages 381–392, 2004.

24 J. G. Rosenstein. *Linear orderings*. Academic Press, New York, 1982.

25 M. P. Schützenberger. On finite monoids having only trivial subgroups. *Inform. Control*, 8:190–194, 1965.

26 W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume III, pages 389–455. Springer-Verlag, 1997.

Determinizing Discounted-Sum Automata*

Udi Boker[1,2] and Thomas A. Henzinger[2]

1 **Hebrew University of Jerusalem**
2 **IST Austria**

Abstract

A discounted-sum automaton (NDA) is a nondeterministic finite automaton with edge weights, which values a run by the discounted sum of visited edge weights. More precisely, the weight in the i-th position of the run is divided by λ^i, where the discount factor λ is a fixed rational number greater than 1. Discounted summation is a common and useful measuring scheme, especially for infinite sequences, which reflects the assumption that earlier weights are more important than later weights. Determinizing automata is often essential, for example, in formal verification, where there are polynomial algorithms for comparing two deterministic NDAs, while the equivalence problem for NDAs is not known to be decidable. Unfortunately, however, discounted-sum automata are, in general, not determinizable: it is currently known that for every rational discount factor $1 < \lambda < 2$, there is an NDA with λ (denoted λ-NDA) that cannot be determinized.

We provide positive news, showing that every NDA with an integral factor is determinizable. We also complete the picture by proving that the integers characterize exactly the discount factors that guarantee determinizability: we show that for every rational factor $\lambda \notin \mathbb{N}$, there is a nondeterminizable λ-NDA. Finally, we prove that the class of NDAs with integral discount factors enjoys closure under the algebraic operations min, max, addition, and subtraction, which is not the case for general NDAs nor for deterministic NDAs. This shows that for integral discount factors, the class of NDAs forms an attractive specification formalism in quantitative formal verification. All our results hold equally for automata over finite words and for automata over infinite words.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Discounted-sum automata, determinization, quantitative verification

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.82

1 Introduction

Discounting the influence of future events is a key paradigm in economics and it is studied in game theory (e.g. [12, 1]), Markov decision processes (e.g. [10, 9]), and automata theory (e.g. [5, 8, 2, 3, 4]). Discounted summation formalizes the concept that an immediate reward is better than a potential one in the far-away future, as well as that a potential problem in the future is less troubling than a current one.

A discounted-sum automaton (NDA) is a nondeterministic automaton with rational weights on the transitions, where the value of a run is the discounted summation of the weights along it. Each automaton has a fixed discount-factor λ, which is a rational number bigger than 1, and the weight in the ith position of a run is divided by λ^i. The value of a word is the minimal value of the automaton runs on it. Hence, an NDA realizes a function from words to real numbers. Two automata are equivalent if they realize the same function, namely if they assign the same value to every word.

* The research was supported by the FWF NFN Grant No S11407-N23 (RiSE), the EU STREP Grant COMBEST, the ERC Advanced Grant QUAREM, and the EU NOE Grant ArtistDesign.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 82–96

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

Discounted summation is of special interest for automata over infinite words. There are two common ways to adjust standard summation for handling infinite sequences: discounting and limit-averaging. The latter, which relates to the input suffixes, has been studied a lot in mean-payoff games and, more recently, in limit-average automata [2, 6]; the former, which relates more to the input prefixes, has received comparatively little attention.

Automata are widely used in formal verification, for which automata comparison is fundamental. Specifically, one usually considers the following three questions, ordered from the most difficult one to the simplest one: general comparison (language inclusion), universality, and emptiness. In the Boolean setting, where automata assign Boolean values to the input words, the three questions, with respect to automata $\mathcal{A}$ and $\mathcal{B}$, are whether $\mathcal{A} \subseteq \mathcal{B}$, $\mathcal{A} = \mathit{True}$, and $\mathcal{A} = \mathit{False}$. In the quantitative setting, where automata assign numeric values to the input words, the universality and emptiness questions relate to a constant threshold, usually 0. Thus, the three questions are whether $\mathcal{A} \leq \mathcal{B}$, $\mathcal{A} \leq 0$, and $\mathcal{A} \geq 0$.

A central problem with these quantitative automata is that only the emptiness question is known to be solvable. For limit-average automata, the two other questions are undecidable [6]. For NDAs, it is an open question whether universality and comparison are decidable. This is not the case with DDAs, for which all three questions have polynomial solutions [12, 1, 2]. Unfortunately, NDAs cannot, in general, be determinized. It is currently known that for every rational discount-factor $1 < \lambda < 2$, there is a λ-NDA that cannot be determinized [2].

It turns out, quite surprisingly, that discounting by an integral factor forms a "well behaved" class of automata, denoted "integral NDAs", allowing for determinization (Section 3) and closed under the algebraic operations min, max, addition and subtraction (Section 5). The above closure is of special interest, as neither NDAs nor DDAs are closed under the max operation (Theorem 9). Furthermore, the integers, above 1, characterize exactly the set of discount factors that guarantee determinizability (Section 4). That is, for every rational factor $\lambda \notin \mathbb{N}$, there is a non-determinizable λ-NDA.

The discounted summation intuitively makes NDAs more influenced by word-prefixes than by word-suffixes, suggesting that some basic properties are shared between automata over finite words and over infinite words. Indeed, all the above results hold for both models. Yet, the equivalence relation between automata over infinite words is looser than the one on finite words. That is, if two automata are equivalent with respect to finite words then they are also equivalent with respect to infinite words, but not vice versa (Lemma 3).

The above results relate to complete automata; namely, to automata in which every state has at least one transition over every alphabet letter. For incomplete automata or, equivalently, for automata with ∞-weights, no discount factor can guarantee determinization (Section 4.2).

Our determinization procedure, described in Section 3.1, is an extension of the subset construction, keeping a "recoverable-gap" value to each element of the subset. Intuitively, the "gap" of a state q over a finite word u stands for the extra cost of reaching q, compared to the best possible value so far. This extra cost is multiplied, however, by $\lambda^{|u|}$, to reflect the $\lambda^{|u|}$ division in the value-computation of the suffixes. A gap of q over u is "recoverable" if there is a suffix w that "recovers" it, meaning that there is an optimal run over uw that visits q after reading u. Due to the discounting of the future, once a gap is too large, it is obviously not recoverable. Specifically, for every λ, we have that $\sum_{i=0}^{\infty}(\frac{1}{\lambda^i}) = \frac{1}{1-\frac{1}{\lambda}} = \frac{\lambda}{\lambda-1} \leq 2$. Hence, our procedure only keeps gaps that are smaller than twice the maximal difference between the automaton weights.

The determinization procedure may be used for an arbitrary λ-NDA, always providing an equivalent λ-DDA, if terminating. Yet, it is guaranteed to terminate for a λ-NDA with

$\lambda \in \mathbb{N}$, while it might not terminate in the case that $\lambda \in \mathbb{Q} \setminus \mathbb{N}$.

For integral NDAs, the key observation is that there might only be finitely many recoverable gaps (Lemma 2). More precisely, for an integral NDA $\mathcal{A}$, there might be up to m recoverable gaps, where m is the maximal difference between the weights in $\mathcal{A}$, multiplied by the minimal common divider of all weights. Accordingly, our determinization procedure generates a DDA with up to m^n states, where n is the number of states in $\mathcal{A}$. We show that this state blow-up is tight, using a rich alphabet of size exponential in the number of states (Theorem 6). The unavoidable state blow-up for the case that the alphabet size is linear in the number of states is left as an open problem.

For nonintegral NDAs, the key observation is that the recoverable gaps might be arbitrarily dense (Theorem 7). Hence, the bound on the maximal value of the gaps cannot guarantee a finite set of recoverable gaps. Different gaps have, under the appropriate setting, suffixes that distinguish between them, implying that an equivalent deterministic automaton must have a unique state for each recoverable-gap (Lemma 5). Therefore, an automaton that admits infinitely many recoverable gaps cannot be determinized.

It turns out that closure under algebraic operation is also closely related to the question of whether the set of recoverable gaps is finite. Considering the operations of addition, subtraction, minimization, and maximization, the latter is the most problematic one, as the value of a word is defined to be the minimal value of the automaton runs on it. For two NDAs, $\mathcal{A}$ and $\mathcal{B}$, one may try to construct an automaton $\mathcal{C} = \max(\mathcal{A}, \mathcal{B})$, by taking the product of $\mathcal{A}$ and $\mathcal{B}$, while maintaining the recoverable gaps of $\mathcal{A}$'s original states, compared to $\mathcal{B}$'s original states. This approach indeed works for integral NDAs (Theorem 10). Note that determinizability is not enough, as neither NDAs nor DDAs are closed under the max operation. Furthermore, we show, in Theorem 9, that there are two DDAs, $\mathcal{A}$ and $\mathcal{B}$, such that there is no NDA $\mathcal{C}$ with $\mathcal{C} = \max(\mathcal{A}, \mathcal{B})$. For precluding the existence of such a nondeterministic automaton $\mathcal{C}$, we cannot make usage of Lemma 5, and thus use a more involved, "pumping-style", argument with respect to recoverable gaps.

Related work. Weighted automata are often handled as *formal power series*, mapping words to a *semiring* [7]. By this view, the weight of a run is the semiring-multiplication of the transition weights along it, while the weight of a word is the semiring-addition of its possible run weights. For this setting, there are numerous works, including results on determinization [11, 7]. However, discounted-sum automata do not fall into this setting, as discounted summation cannot be described as the multiplication operation of a semiring. The latter is required to have an identity element $\bar{1}$, such that for every element e, $\bar{1}e = e\bar{1} = e$. One can check that discounted summation cannot allow for an identity element, which is, in a sense, the core reason for its different behavior. Formal power series are generalized, in [8], for handling discounted summation. The weight of a run is defined to be a "skewed multiplication" of the weights along it, where this "skewing" corresponds to the discounting operation. Yet, [8] mainly considers the equivalence between recognizable series and rational series, and does not handle automata determinization.

Discounted Markov decision processes (e.g. [10, 9]) and discounted games (e.g. [12, 1]) generalize, in some sense, deterministic discounted-sum automata. The former adds probabilities and the latter allows for two player choices. However, they do not cover nondeterministic automata. One may note that nondeterminism relates to "blind games", in which each player cannot see the other player's moves, whereas in standard games the players have full information on all moves. Indeed, for a discounted-game, one can always compute an optimal strategy [12], while a related question on nondeterministic discounted-sum automata, of whether the value of all words is below 0, is not known to be decidable.

The discounted-sum automata used in [2] are the same as ours, with only syntactic differences – they use the discount-factor λ as a multiplying factor, rather than as a dividing one, and define the value of a word as the maximal value of the automaton runs on it, rather than the minimal one. The definitions are analogous, replacing λ with $\frac{1}{\lambda}$ and multiplying all weights by (-1). In [2], it is shown that for every rational discount-factor $1 < \lambda < 2$, there is a λ-NDA that cannot be determinized. We generalize their proof approach, in Theorem 7, extending the result to every $\lambda \in \mathbb{Q} \setminus \mathbb{N}$.

2 Discounted-Sum Automata

We consider discounted-sum automata with rational weights and rational discount factors over finite and infinite words.

Formally, given an alphabet Σ, a *word* over Σ is a finite or infinite sequence of letters in Σ, with ε for the empty word. We denote the concatenation of a finite word u and a finite or infinite word w by $u \cdot w$, or simply by uw.

A discounted-sum automaton (NDA) is a tuple $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \gamma, \lambda \rangle$ over a finite alphabet Σ, with a finite set of states Q, an initial state $q_{in} \in Q$, a transition function $\delta \subseteq Q \times \Sigma \times Q$, a weight function $\gamma : \delta \to \mathbb{Q}$, and a discount factor $1 < \lambda \in \mathbb{Q}$. We write λ-NDA to denote an NDA with a discount factor λ, for example $\frac{5}{2}$-NDA, and refer to an "integral NDA" when λ in an integer. For an automaton $\mathcal{A}$ and a state q of $\mathcal{A}$, we denote by $\mathcal{A}^q$ the automaton that is identical to $\mathcal{A}$, except for having q as its initial state.

Intuitively, $\{q' \mid (q, \sigma, q') \in \delta\}$ is the set of states that $\mathcal{A}$ may move to when it is in the state q and reads the letter σ. The automaton may have many possible transitions for each state and letter, and hence we say that $\mathcal{A}$ is *nondeterministic*. In the case where for every $q \in Q$ and $\sigma \in \Sigma$, we have that $|\{q' \mid (q, \sigma, q') \in \delta\}| \leq 1$, we say that $\mathcal{A}$ is *deterministic*, denoted DDA.

In the case where for every $q \in Q$ and $\sigma \in \Sigma$, we have that $|\{q' \mid (q, \sigma, q') \in \delta\}| \geq 1$, we say that $\mathcal{A}$ is *complete*. Intuitively, a complete automaton cannot get stuck at some state. In this paper, we only consider complete automata, except for Section 4.2, handling incomplete automata.

A run of an automaton is a sequence of states and letters, $q_0, \sigma_1, q_1, \sigma_2, q_2, \ldots$, such that $q_0 = q_{in}$ and for every i, $(q_i, \sigma_{i+1}, q_{i+1}) \in \delta$. The length of a run, denoted $|r|$, is n for a finite run $r = q_0, \sigma_1, q_1, \ldots, \sigma_n, q_n$, and ∞ for an infinite run.

The value of a run r is $\gamma(r) = \sum_{i=0}^{|r|-1} \frac{\gamma(q_i, \sigma_{i+1}, q_{i+1})}{\lambda^i}$. The value of a word w (finite or infinite) is $\mathcal{A}(w) = \inf\{\gamma(r) \mid r \text{ is a run of } \mathcal{A} \text{ on } w\}$. A run r of $\mathcal{A}$ on a word w is said to be *optimal* if $\gamma(r) = \mathcal{A}(w)$. By the above definitions, an automaton $\mathcal{A}$ over finite words realizes a function from Σ^* to $\mathbb{Q}$ and over infinite words from Σ^ω to $\mathbb{R}$. Two automata, $\mathcal{A}$ and $\mathcal{A}'$, are *equivalent* if they realize the same function.

Next, we provide some specific definitions, to be used in the determinization construction and in the non-determinizability proofs.

The *cost* of reaching a state q of an automaton $\mathcal{A}$ over a finite word u is $\text{cost}(q, u) = \min\{\gamma(r) \mid r \text{ is a run of } \mathcal{A} \text{ on } u \text{ ending in } q\}$, where $\min \emptyset = \infty$. The *gap* of a state q over a finite word u is $\text{gap}(q, u) = \lambda^{|u|}(\text{cost}(q, u) - \mathcal{A}(u))$. Note that when $\mathcal{A}$ operates over infinite words, we interpret $\mathcal{A}(u)$, for a finite word u, as if $\mathcal{A}$ was operating over finite words.

Intuitively, the gap of a state q over a word u stands for the weight that a run starting in q should save, compared to a run starting in u's optimal ending state, in order to make q's path preferable. A gap of a state q over a finite word u is said to be *recoverable* if there is a suffix that makes this path optimal; that is, if there is a word w, such that

$\text{cost}(q,u) + \frac{\mathcal{A}^q(w)}{\lambda^{|u|}} = \mathcal{A}(uw)$. The suffix w should be finite/infinite, depending on whether $\mathcal{A}$ operates over finite/infinite words.

Notes on notation-conventions: The discount factor λ is often used in the literature as a multiplying factor, rather than as a dividing factor, thus taking the role of $\frac{1}{\lambda}$, compared to our definitions. Another convention is to value a word as the maximal value of its possible runs, rather than the minimal value; the two definitions are analogous, and can be interchanged by multiplying all weights by (-1).

3 Determinizability of Integral Discounted-Sum Automata

In this section, we show that all complete NDAs with an integral factor are determinizable. Formally, we provide the following result.

▶ **Theorem 1.** *For every complete λ-NDA $\mathcal{A}$ with an integral factor $\lambda \in \mathbb{N}$, there is an equivalent complete λ-DDA with up to m^n states, where m is the maximal difference between the weights in $\mathcal{A}$, multiplied by the minimal common divider of all weights, and n is the number of states in $\mathcal{A}$.*

Proof. Lemmas 2-4, given in the subsections below, constitute the proof. ◀

Theorem 1 stands for both automata over finite words and over infinite words.

The determinization procedure extends the subset construction, by keeping a recoverable-gap value to each element of the subset. It resembles the determinization procedure of non-discounting sum automata over finite words [11, 7], while having two main differences: the weight-differences between the reachable states is multiplied at every step by λ, and differences that exceed some threshold are removed.

The procedure may be used for an arbitrary λ-NDA, always providing an equivalent λ-DDA, if terminating. It is guaranteed to terminate for a λ-NDA with $\lambda \in \mathbb{N}$, which is not the case for $\lambda \in \mathbb{Q} \setminus \mathbb{N}$.

The state blow-up involved in the construction is shown to be tight for a rich alphabet of size exponential in the number of states (Theorem 6). The unavoidable blow-up for an alphabet of size linear in the number of states is left as an open problem.

We start, in Subsection 3.1, with the determinization procedure, continue, in Subsection 3.2, with its termination and correctness proofs, and conclude, in Subsection 3.3, showing that the involved state blow-up is tight for a rich alphabet.

3.1 The Construction

Consider an NDA $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \gamma, \lambda \rangle$. We inductively construct an equivalent DDA $\mathcal{D} = \langle \Sigma, Q', q'_{in}, \delta', \gamma', \lambda \rangle$. (An example is given in Figure 4.)

Let T be the maximal difference between the weights in $\mathcal{A}$. That is, $T = \max\{|x-y| \,\big|\, x, y \in \text{range}(\gamma)\}$. Since $\sum_{i=0}^{\infty}(\frac{1}{\lambda^i}) = \frac{1}{1-\frac{1}{\lambda}} = \frac{\lambda}{\lambda-1} \leq 2$, we define the set $G = \{v \mid v \in \mathbb{Q} \text{ and } 0 \leq v < 2T\} \cup \{\infty\}$ of possible recoverable-gaps. The ∞ element denotes a non-recoverable gap, and behaves as the standard infinity element in the arithmetic operations that we will be using. Note that our discounted-sum automata do not have infinite weights; it is only used as an internal element of the construction.

A state of $\mathcal{D}$ extends the standard subset construction by assigning a gap to each state of $\mathcal{A}$. That is, for $Q = \{q_1, \ldots, q_n\}$, a state $q' \in Q'$ is a tuple $\langle g_1, \ldots, g_n \rangle$, where $g_h \in G$ for every $1 \leq h \leq n$. Intuitively, the gap g_h of a state q_h stands for the extra cost of reaching q_h, compared to the best possible value so far. This extra cost is multiplied, however, by λ^l,

for a finite run of length l, to reflect the λ^l division in the value-computation of the suffixes. Once a gap is obviously irreducible, by being larger than or equal to $2T$, it is set to be ∞.

In the case that $\lambda \in \mathbb{N}$, the construction only requires finitely many elements of G, as shown in Lemma 2 below, and thus it is guaranteed to terminate.

For simplicity, we assume that $q_{in} = q_1$ and extend γ with $\gamma(\langle q_i, \sigma, q_j \rangle) = \infty$ for every $\langle q_i, \sigma, q_j \rangle \notin \delta$. The initial state of $\mathcal{D}$ is $q'_{in} = \langle 0, \infty, \ldots, \infty \rangle$, meaning that q_{in} is the only relevant state and has a 0 gap.

We inductively build $\mathcal{D}$ via the intermediate automata $\mathcal{D}_i = \langle \Sigma, Q'_i, q'_{in}, \delta'_i, \gamma'_i, \lambda \rangle$. We start with $\mathcal{D}_1$, in which $Q'_1 = \{q'_{in}\}$, $\delta'_1 = \emptyset$ and $\gamma'_1 = \emptyset$, and proceed from $\mathcal{D}_i$ to $\mathcal{D}_{i+1}$, such that $Q'_i \subseteq Q'_{i+1}$, $\delta'_i \subseteq \delta'_{i+1}$ and $\gamma'_i \subseteq \gamma'_{i+1}$. The construction is completed once $\mathcal{D}_i = \mathcal{D}_{i+1}$, finalizing the desired deterministic automaton $\mathcal{D} = \mathcal{D}_i$.

In the induction step, $\mathcal{D}_{i+1}$ extends $\mathcal{D}_i$ by (possibly) adding, for every state $q' = \langle g_1, \ldots, g_n \rangle \in Q'_i$ and letter $\sigma \in \Sigma$, a state q'', a transition $\langle q', \sigma, q'' \rangle$ and a weight $\gamma_{i+1}(\langle q', \sigma, q'' \rangle) = c$, as follows:

- For every $1 \leq h \leq n$, $c_h := \min\{g_j + \gamma(\langle q_j, \sigma, q_h \rangle) \mid 1 \leq j \leq n\}$
- $c := \min_{1 \leq h \leq n} (c_h)$
- For every $1 \leq h \leq n$,
  - $x_h := \lambda(c_h - c)$;
  - if $x_h \geq 2T$ then $x_h := \infty$
- $q'' := \langle x_1, \ldots, x_n \rangle$
- $Q'_{i+1} := Q'_{i+1} \cup q''$
- $\delta'_{i+1} := \delta'_{i+1} \cup \langle q', \sigma, q'' \rangle$
- $\gamma'_{i+1}(\langle q', \sigma, q'' \rangle) = c$

3.2 Termination and Correctness

We prove below that the above procedure always terminates for a discount factor $\lambda \in \mathbb{N}$, while generating an automaton that is equivalent to the original one. We start with the termination proof.

▶ **Lemma 2.** *The above determinization procedure always terminates for a complete integral λ-NDA $\mathcal{A}$. The resulting deterministic automaton has up to m^n states, where m is the maximal difference between the weights in $\mathcal{A}$, multiplied by the minimal common divider of all weights, and n is the number of states in $\mathcal{A}$.*

Proof. The induction step of the construction, extending $\mathcal{D}_i$ to $\mathcal{D}_{i+1}$, only depends on $\mathcal{A}$, Σ and Q'_i. Furthermore, for every $i \geq 0$, we have that $Q'_i \subseteq Q'_{i+1}$. Thus, for showing the termination of the construction, it is enough to show that there is a general bound on the size of the sets Q'_i. We do it by showing that the inner values, $g_1, \ldots, g_n$, of every state q' of every set Q'_i are from the finite set $\bar{G}$, defined below.

Let $d \in \mathbb{N}$ be the minimal common divider of the weights in $\mathcal{A}$, and let $m \in \mathbb{N}$ be the maximal difference between the weights, multiplied by d. That is, $m = d \times \max\{|x-y| \mid x, y \in \text{range}(\gamma)\}$. We define the set $\bar{G} = \{\frac{\lambda c}{d} \mid \frac{2m}{\lambda} > c \in \mathbb{N}\} \cup \{\infty\}$

We start with Q'_1, which satisfies the property that the inner values, $g_1, \ldots, g_n$, of every state $q' \in Q'_1$ are from $\bar{G}$, as $Q'_1 = \{\langle 0, \infty, \ldots, \infty \rangle\}$. We proceed by induction on the construction steps, assuming that Q'_i satisfies the property. By the construction, an inner value of a state q'' of Q'_{i+1} is derived by four operations on elements of $\bar{G}$: addition, subtraction ($x - y$, where $x \geq y$), multiplication by $\lambda \in \mathbb{N}$, and minimization.

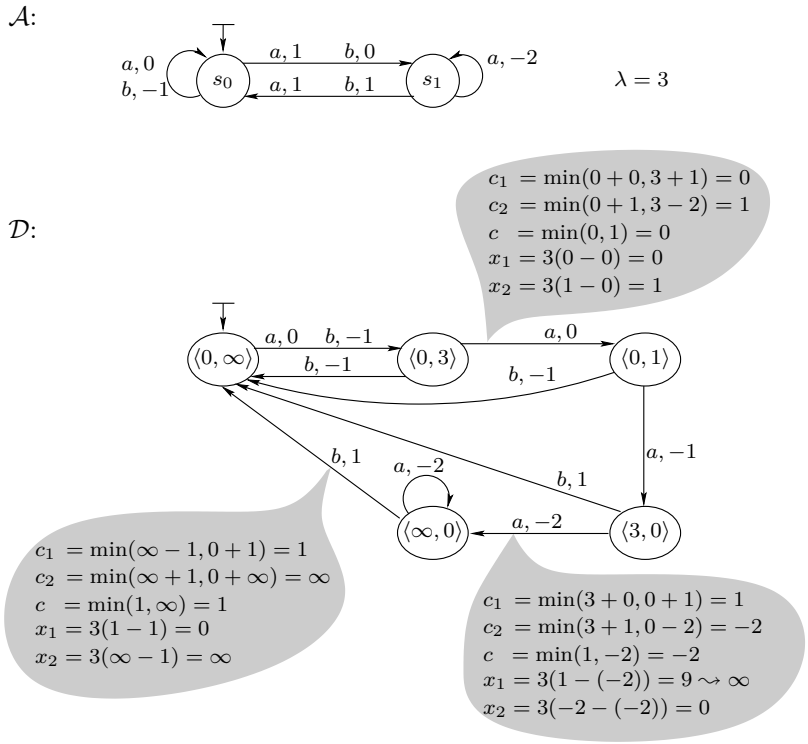


Figure 1 Determinizing the 3-NDA $\mathcal{A}$ into the 3-DDA $\mathcal{D}$. The gray bubbles detail some of the intermediate calculations of the determinization procedure.

One may verify that applying these four operations on ∞ and numbers of the form $\frac{\lambda c}{d}$, where $\lambda, c \in \mathbb{N}$, results in ∞ or in a number $\frac{v}{d}$, where $v \in \mathbb{N}$. Since the last operation in calculating an inner value of q'' is multiplication by λ, we have that v is divided by λ. Once an inner value exceeds $\frac{2m}{d}$, it is replaced with ∞. Hence, all the inner values are in $\bar{G}$.

Having up to m possible values to the elements of an n-tuple, provides the m^n upper bound for the state space of the resulting deterministic automaton. ◀

Before proceeding to the correctness proof, we show that equivalence of automata over finite words implies their equivalence over infinite words. Note that the converse need not hold.

▶ **Lemma 3.** *If two NDAs, $\mathcal{A}$ and $\mathcal{B}$, are equivalent with respect to finite words then they are also equivalent with respect to infinite words. The converse need not hold.*

Proof. Assume, by contradiction, two NDAs, $\mathcal{A}$ and $\mathcal{B}$, that are equivalent with respect to finite words and not equivalent with respect to infinite words. Then there is an infinite word w and a constant number $c \neq 0$, such that $\mathcal{A}(w) - \mathcal{B}(w) = c$. Let m be the maximal difference between a weight in $\mathcal{A}$ and a weight in $\mathcal{B}$. Since for every $1 < \lambda$, $\sum_{i=0}^{\infty}(\frac{1}{\lambda^i}) = \frac{1}{1-\frac{1}{\lambda}} = \frac{\lambda}{\lambda-1} \leq 2$, it follows that the difference between the values that $\mathcal{A}$ and $\mathcal{B}$ assign to any word is smaller or equal to $2m$. Hence, the difference between the values of their runs on suffixes of w, starting at a position p, is smaller or equal to $\frac{2m}{\lambda^p}$.

Now, since $\mathcal{A}$ and $\mathcal{B}$ are equivalent over finite words, it follows that they have equally-valued optimal runs over every prefix of w. Thus, after a long enough prefix, of length p such that $\frac{2m}{\lambda^p} < c$, the difference between the values of $\mathcal{A}$'s and $\mathcal{B}$'s optimal runs on w must be smaller than c, leading to a contradiction.

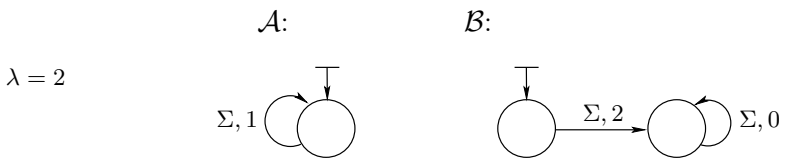


Figure 2 The automata $\mathcal{A}$ and $\mathcal{B}$ are equivalent with respect to infinite words, while not equivalent with respect to finite words.

A counter example for the converse is provided in Figure 2. ◀

We proceed with the correctness proof. By Lemma 3, it is enough to prove the correctness for automata over finite words.

Note that the correctness holds for arbitrary discount factors, not only for integral ones. For the latter, the determinization procedure is guaranteed to terminate (Lemma 2), which is not the case in general. Yet, in all cases that the procedure terminates, it is guaranteed to be correct.

▶ **Lemma 4.** *Consider a λ-NDA $\mathcal{A}$ over Σ^* and a DDA $\mathcal{D}$, constructed from $\mathcal{A}$ as above. Then, for every $w \in \Sigma^*$, $\mathcal{A}(w) = \mathcal{D}(w)$.*

Proof. Consider an NDA $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \gamma, \lambda \rangle$ and the DDA $\mathcal{D} = \langle \Sigma, Q', q'_{in}, \delta', \gamma', \lambda \rangle$ constructed from $\mathcal{A}$ as above. Let T be the maximal difference between the weights in $\mathcal{A}$. That is, $T = \max\{|x - y| \mid x, y \in \text{range}(\gamma)\}$.

For a word w, let $q'_w = \langle g_1, \ldots, g_n \rangle \in Q'$ be the last state of $\mathcal{D}$'s run on w. We show by induction on the length of the input word w that:

- For every $1 \leq h \leq n$, $g_h = \text{gap}(q_h, w)$ if $\text{gap}(q_h, w) < 2T$ and ∞ otherwise.
- $\mathcal{A}(w) = \mathcal{D}(w)$.

The assumptions obviously hold for the initial step, where w is the empty word. As for the induction step, we assume they hold for w and show that for every $\sigma \in \Sigma$, they hold for $w{\cdot}\sigma$. Let $q'_{w\sigma} = \langle x_1, \ldots, x_n \rangle \in Q'$ be the last state of $\mathcal{D}$'s run on $w{\cdot}\sigma$.

For every $1 \leq h \leq n$, as long as $g_h < 2T$, the value that the determinization-construction assigns to x_h, as well as the weight that is set on the transition from q'_w to $q'_{w\sigma}$, directly follows the gap definitions, and accordingly satisfy the required properties. Therefore, it is left to show that if $g_h \geq 2T$ then $\text{gap}(q_h, w\sigma) \geq 2T$. Indeed, $\text{gap}(q_h, w\sigma) = \lambda^{i+1}(\text{cost}(q_h, w\sigma) - \mathcal{A}(w\sigma)) > \lambda^{i+1}(\text{cost}(q_h, w) - \mathcal{A}(w) - (\frac{1}{\lambda}^i)T) > \lambda(2T - T) = \lambda(T) \geq 2T$.

◀

3.3 State Complexity

For an integral NDA $\mathcal{A}$, the deterministic automaton constructed as in Subsection 3.1 has up to m^n states, where m is the maximal difference between the weights in $\mathcal{A}$, multiplied by the minimal common divider of all weights, and n is the number of states in $\mathcal{A}$ (Lemma 2).

We show below that the above state blow-up is asymptotically tight, using a rich alphabet of size in $O(m^n)$. For an alphabet of size linear in m and n, the unavoidable state blow-up is left as an open problem.

A family of automata $\mathcal{A}_{m,n}$, with which we provide the lower bound, is illustrated in Figure 3. Intuitively, the rich alphabet allows to set every gap in $\{0, 1, 2, \ldots, m+1\}$ to each of the n states. Two different gaps have, under the appropriate setting, two suffixes that

distinguish between them. Hence, an equivalent deterministic automaton must have a unique state for each recoverable-gap, yielding at least m^n states.

We start by providing a sufficient condition, under which two different gaps must be associated with two different states of a deterministic automaton. The lemma below generalizes an argument given in [2].

▶ **Lemma 5.** *Consider an NDA $\mathcal{A}$ for which there is an equivalent DDA $\mathcal{D}$. If there is a state q of $\mathcal{A}$, finite words u and u', and words w and z, such that:*

i. *$\mathcal{A}$ has runs on u and on u' ending in q;*
ii. $\text{gap}(q,u) \neq \text{gap}(q,u')$*;*
iii. *The gaps of q over both u and u' are recoverable with w, that is, $\mathcal{A}(uw) = \text{cost}(q,u) + \frac{\mathcal{A}^q(w)}{\lambda^{|u|}}$ and $\mathcal{A}(u'w) = \text{cost}(q,u') + \frac{\mathcal{A}^q(w)}{\lambda^{|u'|}}$; and*
iv. *$\mathcal{A}$ is indifferent to concatenating z to u and to u', that is $\mathcal{A}(uz) = \mathcal{A}(u)$ and $\mathcal{A}(u'z) = \mathcal{A}(u')$*

then the runs of $\mathcal{D}$ on u and on u' end in different states.

The words w and z should be finite for automata over finite words and infinite for automata over infinite words. In the former case, z is redundant as it can always be ε.

Proof. Consider the above setting. Then, we have that $\mathcal{A}(uw) - \mathcal{A}(uz) = \frac{\text{gap}(q,u)+\mathcal{A}^q(w)}{\lambda^{|u|}}$ and $\mathcal{A}(u'w) - \mathcal{A}(u'z) = \frac{\text{gap}(q,u')+\mathcal{A}^q(w)}{\lambda^{|u'|}}$. Thus,

$$(I)\ \ \text{gap}(q,u) = \lambda^{|u|}[\mathcal{A}(uw) - \mathcal{A}(uz)] - \mathcal{A}^q(w); \quad \text{gap}(q,u') = \lambda^{|u'|}[\mathcal{A}(u'w) - \mathcal{A}(u'z)] - \mathcal{A}^q(w)$$

Now, assume, by contradiction, a single state p of $\mathcal{D}$ in which the runs of $\mathcal{D}$ on both u and u' end. Then, we have that

$$(II)\ \ \mathcal{D}(uw) - \mathcal{D}(uz) = \frac{\mathcal{D}^p(w)}{\lambda^{|u|}}; \quad \mathcal{D}(u'w) - \mathcal{D}(u'z) = \frac{\mathcal{D}^p(w)}{\lambda^{|u'|}}$$

Since $\mathcal{A}$ and $\mathcal{D}$ are equivalent, we may replace between $[\mathcal{A}(uw) - \mathcal{A}(uz)]$ and $[\mathcal{D}(uw) - \mathcal{D}(uz)]$ as well as between $[\mathcal{A}(u'w) - \mathcal{A}(uz')]$ and $[\mathcal{D}(u'w) - \mathcal{D}(u'z)]$. Making the replacements in equations (I) above, we get:

$$(I\&II)\ \ \text{gap}(q,u) = \lambda^{|u|}\frac{\mathcal{D}^p(w)}{\lambda^{|u|}} - \mathcal{A}^q(w); \quad \text{gap}(q,u') = \lambda^{|u'|}\frac{\mathcal{D}^p(w)}{\lambda^{|u'|}} - \mathcal{A}^q(w)$$

Therefore, $\text{gap}(q,u) = \text{gap}(q,u')$, leading to a contradiction.

◀

We continue with the tightness proof.

▶ **Theorem 6.** *For every $\lambda, m, n \in \mathbb{N}$, there is a complete λ-NDA with $n+2$ states and weights in $\{-\lambda m, -\lambda m + 1, \ldots, -1, 0, 1\}$, such that every equivalent DDA has at least m^n states.*

Proof. For every $\lambda, m, n \in \mathbb{N}$, we define the NDA $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \gamma, \lambda \rangle$, illustrated in Figure 3 as $\mathcal{A}_{m,n}$, where:

- $\Sigma = \{\langle v_1, \ldots v_n \rangle \mid \text{ for every } 1 \leq i \leq n, v_i \in \{-\lambda m, -\lambda m + 1, \ldots, -1, 0, 1\}\}$
- $Q = \{q_{in}, q_0, q_1, \ldots, q_n\}$
- $\delta = \{\langle q_{in}, \sigma, q_i \rangle, \langle q_i, \sigma, q_i \rangle \mid 0 \leq i \leq n \text{ and } \sigma \in \Sigma\}$
- For every $\sigma = \langle v_1, \ldots v_n \rangle \in \Sigma$ and $0 \leq i \leq n$: $\gamma(\langle q_{in}, \sigma, q_0 \rangle) = 0$, $\gamma(\langle q_{in}, \sigma, q_i \rangle) = 0$, $\gamma(\langle q_0, \sigma, q_0 \rangle) = 0$ and $\gamma(\langle q_i, \sigma, q_i \rangle) = v_i$

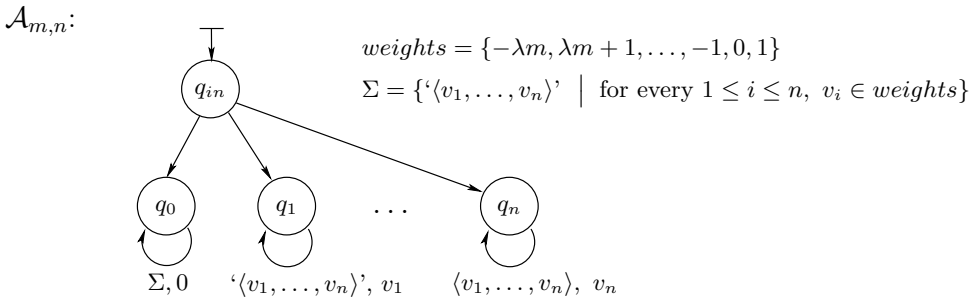


Figure 3 The family of integral NDAs, where for every m and n, a deterministic automaton equivalent to $\mathcal{A}_{m,n}$ must have at least m^n states.

Note that, for simplicity, we define the alphabet letters of Σ as tuples of numbers.

Consider a DDA $\mathcal{D}$ equivalent to $\mathcal{A}$. We will show that there is a surjective mapping between $\mathcal{D}$'s states and the set of vectors $V = \{\langle g_1, \ldots, g_n\rangle \mid \text{for every } 1 \leq i \leq n, 1 \leq g_i \leq m\}$.

We call an n-vector of gaps, $G = \langle g_1, \ldots, g_n\rangle$, a *combined-gap*, specifying the gaps of $q_1, \ldots, q_n$, respectively. Due to the rich alphabet, for every combined-gap $G \in V$, there is a finite word u_G, such that for every $1 \leq i \leq n$, $\text{gap}(q_i, u_G) = g_i$.

Every two different combined gaps, G and G', are different in at least one dimension j of their n-vectors. Thus, $\mathcal{A}$ satisfies the conditions of Lemma 5, by having $u = u_G$, $u' = u_{G'}$, $z = \text{`}\langle 0, \ldots, 0\rangle\text{'}^\omega$, and $w = \text{`}\langle 0, \ldots 0, -\lambda m, 0, \ldots 0\rangle\text{'}^\omega$, where the repeated word in w has 0 in all dimensions except for $-\lambda m$ in the j's dimension. Hence, $\mathcal{A}$ has two different states corresponding to each two different vectors in V, and we are done.

◀

4 Nondeterminizability of Nonintegral Discounted-Sum Automata

The discount-factor λ plays a key role in the question of whether a complete λ-NDA is determinizable. In Section 3, we have shown that an integral factor guarantees the automaton's determinizabilty. In Subsection 4.1 below, we show the converse for every nonintegral factor.

In the whole paper, except for Subsection 4.2 below, we only consider complete automata. In Subsection 4.2, we show that once allowing incomplete automata or, equivalently, adding infinite weights, there is a non-determinizable automaton for every discount-factor λ, including integral ones.

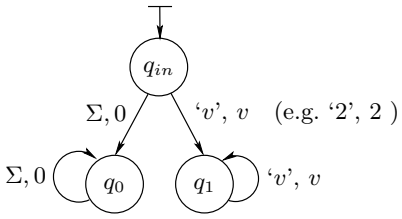
4.1 Complete Automata

We show below that for every noninntegral discount factor λ, there is a complete λ-NDA that cannot be determinized. The proof generalizes the approach taken in [2], where the case of $1 < \lambda < 2$ was handled.

Intuitively, for a discount factor that is not a whole number, a nondeterministic automaton might have arbitrarily dense recoverable-gaps. Two different gaps have, under the appropriate setting, two suffixes that distinguish between them (Lemma 5). Hence, an equivalent deterministic automaton must have a unique state for each recoverable-gap, which is impossible for infinitely many gaps.

▶ **Theorem 7.** *For every nonintegral discount factor λ, there is a complete λ-NDA for which there is no equivalent DDA (with any discount factor).*

$\mathcal{A}$:



$\lambda = \frac{5}{2}$
$\Sigma = \{$ '-5', '-4', '-2', '0', '2' $\}$

Figure 4 The non-determinizable $\frac{5}{2}$-NDA $\mathcal{A}$.

Proof. For every $1 < \lambda \in \mathbb{Q} \setminus \mathbb{N}$, we define a complete λ-NDA $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \gamma, \lambda \rangle$ and show that $\mathcal{A}$ is not determinizable. Let $\lambda = \frac{h}{k}$, where h and k are mutually prime, and define:

- $\Sigma = \{-jk \mid j \in \mathbb{N} \text{ and } jk < h\} \cup \{-h, k\}$
- $Q = \{q_{in}, q_1, q_2\}$
- $\delta = \{\langle q_{in}, \sigma, q_1\rangle, \langle q_{in}, \sigma, q_2\rangle, \langle q_1, \sigma, q_1\rangle, \langle q_2, \sigma, q_2\rangle \mid \sigma \in \Sigma\}$
- For every $\sigma \in \Sigma$ and $q \in Q$: $\gamma(\langle q, \sigma, q_1\rangle) = 0$ and $\gamma(\langle q, \sigma, q_2\rangle) = \sigma$

Note that, for simplicity, we define the alphabet letters of Σ as numbers. The NDA $\mathcal{A}$ for $\lambda = \frac{5}{2}$ is illustrated in Figure 1.

We show that $\mathcal{A}$ cannot be determinized by providing an infinite word w, such that q_2 has a unique recoverable gap for each of w's prefixes. By Lemma 5, such a word w implies that $\mathcal{A}$ cannot be determinized.

We inductively define w, denoting its prefix of of length i by w_i, as follows: the first letter is k and the $i+1$'s letter is '$-jk$', such that $0 \leq \text{gap}(q_2, w_i)\frac{h}{k} - jk \leq k$. Intuitively, each letter is chosen to almost compensate on the gap generated so far, by having the same value as the gap up to a difference of k.

We show that w has the required properties, by the following steps:

1. The word w is infinite and q_2 has a recoverable-gap for each of its prefixes.
2. There is no prefix of w for which q_2's gap is 0.
3. There are no two different prefixes of w for which q_2 has the same gap.

Indeed:

1. Since $\gamma(\langle q_2, -h, q_2\rangle) = -h$, a gap g of q_2 is obviously recoverable if $g \leq h$. We show by induction on the length of w's prefixes that for every $i \geq 1$, we have that $\text{gap}(q_2, w_i) \leq h$. It obviously holds for the initial step, as $w_1 =$'k' and $\text{gap}(q_2, w_1) = k\frac{h}{k} = h$. Assuming that it holds for the i's prefix, we can choose the $i+1$'s letter to be some '$-jk$' $\in \Sigma$, such that $0 \leq \text{gap}(w_i) - jk \leq k$. Hence, we get that $\text{gap}(w_{i+1}) = (\text{gap}(w_i) - jk)\frac{h}{k} \leq k$.
2. Assume, by contradiction, a prefix of w of length $n+1$ whose recoverable-gap is 0. We have then that:
$$(((k\frac{h}{k} - j_1k)\frac{h}{k} - j_2k)\frac{h}{k} \ldots - j_nk)\frac{h}{k} = 0$$
for some $j_1, \ldots, j_n \in \mathbb{N}$. Simplifying the equation, we get that
$$\frac{h^n - j_1kh^{n-1} - j_2k^2h^{n-2} - \ldots - j_nk^n}{k^{n-1}} = 0$$
Therefore, $h^n = j_1kh^{n-1} + \ldots + j_nk^n$. Now, since k divides $j_1kh^{n-1} + \ldots + j_nk^n$, it follows that k divides h^n, which leads to a contradiction, as h and k are mutually prime.
3. Assume, by contradiction, that q_2 has the same gap x for two prefixes, $n \geq 1$ steps apart. We have then that:
$$((((x - j_1k)\frac{h}{k} - j_2k)\frac{h}{k} - j_3k)\frac{h}{k} \ldots - j_nk)\frac{h}{k} = x$$

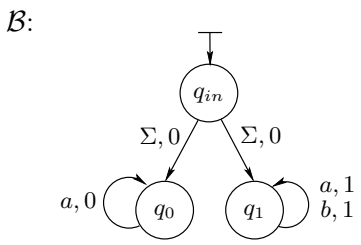


Figure 5 The incomplete automaton $\mathcal{B}$ is not determinizable with respect to any discount-factor.

for some $j_1, \ldots, j_n \in \mathbb{N}$. Simplifying the equation, we get that

$$\frac{xh^n - j_1 k h^n - j_2 k^2 h^{n-1} - \ldots - j_n k^n h}{k^n} = x$$

Thus,

$$xh^n - xk^n = j_1 k h^n + j_2 k^2 h^{n-1} + \ldots + j_n k^n h$$

Hence, k divides $x(h^n - k^n)$. Now, since there is no prefix for which q_2 has a zero gap, it follows that k does not divide x. (Otherwise, q_2 would have had a zero gap for the prefix right after the one with x). Therefore, k divides $h^n - k^n$. But, since k divides k^n, it follows that k also divides h^n, which leads to a contradiction.

◀

4.2 Incomplete Automata

Once considering incomplete automata or, equivalently, automata with ∞-weights, no discount factor can guarantee determinization. The reason is that there is no threshold above which a gap becomes irrecoverable – no matter how (finitely) bad some path is, it might eventually be essential, in the case that the other paths get stuck.

Formally:

▶ **Theorem 8.** *For every rational discount factor λ, there is an incomplete λ-NDA for which there is no equivalent DDA (with any discount factor).*

Proof. Consider the incomplete automaton $\mathcal{B}$ presented in Figure 5 with a discount factor $\lambda \in \mathbb{Q}$.

For every $n \in \mathbb{N}$, we have that $\text{gap}(q_2, a^n) = \sum_{i=0}^{n} \lambda^i$. Since q_1 has no transition for the letter b, it follows that all these gaps are recoverable. Hence, for every $i, j \in \mathbb{N}$ such that $i \neq j$, we satisfy the conditions of Lemma 5 with $u = a^i$, $u' = a^j$, $z = a^\omega$ and $w = b^\omega$ (for automata over finite words, $z = \varepsilon$ and $w = b$). Therefore, an equivalent deterministic automaton must have infinitely many states, precluding its existence. ◀

5 Closure Properties

Discounted-sum automata realize a function from words to numbers. Hence, one may wish to consider their closure under arithmetic operations. The operations are either between two automata, having the same discount factor, as minimization and addition, or between an automaton and a scalar, as multiplication by a positive rational number c.

We consider the class of complete NDAs, as well as two of its subclasses: DDAs and integral NDAs.

Class ╲ Operation	min	max	+	−	$\times c \geq 0$	$\times(-1)$
NDAs	✓	✗	✓	✗	✓	✗
DDAs	✗		✓			
Integral NDAs	✓					

Table 1 Closure of discounted-sum automata under arithmetic operations.

The closure properties, summarized in Table 1, turn out to be the same for automata over finite words and over infinite-words. By arguments similar to those of Lemma 3's proof, it is enough to prove the positive results with respect to automata over finite words and the negative results with respect to automata over infinite words.

Some of the positive results are straightforward, as follows.

- Nondeterministic: Minimization is achieved by taking the union of the input automata, addition by taking the product of the input automata and adding the corresponding weights, and multiplication by a positive scalar c is achieved by multiplying all weights by c.
- Deterministic: Addition/subtraction is achieved by taking the product of the input automata and adding/subtracting the corresponding weights. Multiplication by (positive or negative) scalar c is achieved by multiplying all weights by c.
- Discount factor $\in \mathbb{N}$: Since these automata can always be determinized, they obviously enjoy the closure properties of both the deterministic and non-deterministic classes.

All the negative results can be reduced to the max operation, as follows. Closure under subtraction implies closure under (-1)-multiplication, by subtracting the given automaton from a constant 0 automaton. For nondeterministic automata, closure under (-1)-multiplication implies closure under the max operation, by multiplying the original automata by (-1) and taking their minimum. As for deterministic automata, closure under the min and max operations are reducible to each other due to the closure under (-1)-multiplication.

It is left to show the results with respect to the max operation. We start with the classes of deterministic and nondeterministic automata.

▶ **Theorem 9.** *NDAs and DDAs are not closed under the* max *operation.*

Proof. We prove a stronger claim, showing that there are two DDAs, $\mathcal{A}$ and $\mathcal{B}$, defined in Figure 6, for which there is no NDA equivalent to $\max(\mathcal{A}, \mathcal{B})$. Intuitively, we show that the recoverable-gap between $\mathcal{A}$ and $\mathcal{B}$ can be arbitrarily small, and therefore, by pumping-arguments, an NDA for $\max(\mathcal{A}, \mathcal{B})$ cannot be of a finite size.

Assume, by contradiction, an NDA $\mathcal{C}$ with n states equivalent to $\max(\mathcal{A}, \mathcal{B})$. The value of $\mathcal{A}$ over every word is obviously 0. Thus, for every infinite word w, $\mathcal{C}(w) = \mathcal{B}(w)$ if $\mathcal{B}(w) > 0$ and 0 otherwise.

For a finite word u, we shall refer to $\lambda^{|u|}\mathcal{B}(u)$ as the *gap* of $\mathcal{B}$ over u, denoted $\text{gap}(\mathcal{B}, u)$. Intuitively, this gap stands for the weight that $\mathcal{B}$ should save over a suffix z for having a negative value over the whole word. That is, $\mathcal{B}(uz) < 0$ if and only if $\mathcal{B}(z) < -\text{gap}(\mathcal{B}, u)$. Within this proof, λ is fixed to $\frac{5}{2}$.

A key observation is that the gap of $\mathcal{B}$ can be arbitrarily small. Specifically, we show that for every natural numbers $k \geq 3$ and $j \leq \lceil \frac{2^k}{5} \rceil$, there is a finite word $u_{j,k}$ such that $\text{gap}(\mathcal{B}, u_{j,k}) = \frac{5j}{2^k}$. It goes by induction on k. For $k = 3$, it holds with $u_{0,3} =$'0', $u_{1,3} =$'$\frac{2}{5}$' '$-\frac{1}{2}$' '-1', and $u_{2,3} =$'$\frac{2}{5}$' '$-\frac{1}{2}$'. As for the induction step, consider a number $0 \leq j \leq \lceil \frac{2^{k+1}}{5} \rceil$.

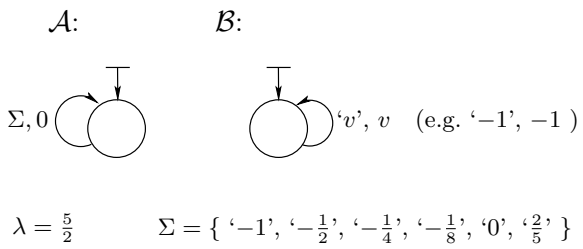


Figure 6 The DDAs $\mathcal{A}$ and $\mathcal{B}$, for which there is no NDA equivalent to $\max(\mathcal{A}, \mathcal{B})$.

One may verify that multiplying 2^k by each of $1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}$, and 0, provides a different reminder when divided by 5. Hence, there is a number $v \in \{-1, -\frac{1}{2}, -\frac{1}{4}, -\frac{1}{8}, 0\}$ and a natural number $j' \leq \lceil \frac{2^k}{5} \rceil$ such that $j' = \frac{j - v2^k}{5}$. Thus, we can have, by the induction assumption, the required word $u_{j,k+1}$, by $u_{j,k+1} = u_{j',k}$'v' , as $\mathrm{gap}(\mathcal{B}, u_{j,k+1}) = \frac{5}{2}(\mathrm{gap}(\mathcal{B}, u_{j',k}) + v) = \frac{5}{2}(\frac{5j'}{2^k} + v) = \frac{5}{2}(\frac{j - v2^k}{2^k} + v) = \frac{5j}{2^{k+1}}$.

By the above observation, there is a finite word u, such that $\frac{1}{\lambda^{2n}} < \mathrm{gap}(\mathcal{B}, u) < \frac{1}{\lambda^n}$. We define the infinite word $w = u$'0'^{n}'-1''0'$^\omega$. Since a 0-weighted letter multiplies the gap by λ, we get that $0 < \mathrm{gap}(\mathcal{B}, u$'0'$^n) < 1$, and therefore $\mathcal{B}(w) < 0$ and $\mathcal{C}(w) = 0$.

Let r be an optimal run of $\mathcal{C}$ on z. Since $\mathcal{C}$ has only n states, there is a state q of $\mathcal{C}$ and two positions $|u| < p_1 < p_2 < |u| + n$, such that r visits q on both p_1 and p_2. Let w_1 and w_2 be the prefixes of w of lengths p_1 and p_2, respectively. Let z be the suffix of w after w_2, that is $w = w_2 z$. Let r_1, r_2 and r_z be the portions of r on w_1, w_2 and z, respectively.

Let v_1 and v_2 be the values of r_1 and r_2, respectively, and define $g_1 = \lambda^{p1} v_1$ and $g_2 = \lambda^{p2} v_2$. Let v_z be the value of a run equivalent to r_z. Since the value of r is 0, we have that $v_2 + \frac{v_z}{\lambda^{p2}} = 0$, and therefore, $v_z = -g_2$.

We shall reach a contradiction by showing that $g_1 \not< g_2$, $g_1 \not> g_2$, and $g_1 \neq g_2$. Indeed:

- If $g_1 < g_2$ then there is a run $r' = r_1 r_z$ of $\mathcal{C}$ on the word $w' = w_1 z$, whose value is $v_1 + \frac{v_z}{\lambda^{p2}} = \frac{g_1 + v_z}{\lambda^{p2}}$. However, since $g_1 < g_2 = -v_z$, it follows that the value of $\mathcal{C}$ on w' is negative, which leads to a contradiction.
- If $g_1 > g_2$ then there is a negative-valued run of $\mathcal{C}$ on the word w_1'0'$^{2(p_2 - p_1)} z$, analogously to the previous case.
- If $g_1 = g_2$ then there is a 0-valued run of $\mathcal{C}$ on the word $w' = w_1$'0'$^{2n} z$, however $\mathcal{B}(w') > 0$, leading to a contradiction.

◀

We continue with the class of automata with an integral factor.

▶ **Theorem 10.** *For every $\lambda \in \mathbb{N}$, the class of λ-NDAs is closed under the* max *operation.*

Proof. Consider a discount-factor $1 < \lambda \in \mathbb{N}$ and two λ-NDAs, $\mathcal{A}$ and $\mathcal{B}$. By Theorem 1, $\mathcal{A}$ and $\mathcal{B}$ can be determinized to equivalent λ-DDAs. Thus, we may only consider deterministic automata. Since deterministic automata are closed under (-1)-multiplication, we may also consider the min operation rather than the max operation.

The construction of a DDA $\mathcal{C}$ equivalent to $\min(\mathcal{A}, \mathcal{B})$ is analogous to the determinization construction of Section 3.1, with the difference of extending automata-product rather than the subset-construction. Namely, we iteratively construct the product of $\mathcal{A}$ and $\mathcal{B}$, where a state of $\mathcal{C}$ contains a state of $\mathcal{A}$ and a state of $\mathcal{B}$, together with their recoverable-gaps. That is, for a state p of $\mathcal{A}$ and a state q of $\mathcal{B}$, a state c of $\mathcal{C}$ is of the form $c = \langle\langle p, g_p\rangle, \langle q, g_q\rangle\rangle$. When $\mathcal{A}$ and $\mathcal{B}$ read a finite word u and reach the states p and q, respectively, we have that

$g_p = \lambda^{|u|}(\mathcal{A}(u) - \min(\mathcal{A}(u), \mathcal{B}(u)))$ and $g_q = \lambda^{|u|}(\mathcal{B}(u) - \min(\mathcal{A}(u), \mathcal{B}(u)))$. Once a gap is too large, meaning that it is bigger than twice the maximal difference between a weight in $\mathcal{A}$ and a weight in $\mathcal{B}$, it is changed to ∞.

The termination and correctness proofs of the above construction are analogous to the proofs of Lemmas 2 and 4. ◀

6 Conclusions

Recently, there has been a considerable effort to extend formal verification from the Boolean setting to a quantitative one. Automata theory plays a key role in formal verification, and therefore quantitative automata, mainly limit-average automata and discounted-sum automata, have a central role in quantitative formal verification. Yet, a bothering problem is that among the basic automata questions underlying a verification task, namely emptiness, universality, and inclusion, only emptiness is known to be solvable for these automata. The other questions are either undecidable, with limit-average automata, or not known to be decidable, with discounted-sum automata.

We showed that discounted-sum automata with an integral factor form a robust class, having algorithms for all the above questions, closed under natural composition relations, as min, max, addition and subtraction, and always allowing for determinization. Hence, we find this class of integral discounted-sum automata a promising direction in the development of formal quantitative verification.

References

1 D. Andersson. An improved algorithm for discounted payoff games. In *ESSLLI Student Session*, pages 91–98, 2006.

2 K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. In *Proc. of CSL*, LNCS 5213, pages 385–400. Springer, 2008.

3 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Alternating weighted automata. In *FCT*, volume 5699 of *LNCS*, pages 3–13, 2009.

4 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Expressiveness and closure properties for quantitative languages. *Logical Methods in Computer Science*, 6(3), 2010.

5 Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Discounting the future in systems theory. In *ICALP*, volume 2719 of *LNCS*, pages 1022–1037, 2003.

6 Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Torunczyk. Energy and mean-payoff games with imperfect information. In *CSL*, pages 260–274, 2010.

7 M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Springer Publishing Company, Incorporated, 2009.

8 Manfred Droste and Dietrich Kuske. Skew and infinitary formal power series. *Theor. Comput. Sci.*, 366(3):199–227, 2006.

9 Hugo Gimbert and Wieslaw Zielonka. Limits of multi-discounted markov decision processes. In *LICS*, pages 89–98. IEEE Computer Society, 2007.

10 Omid Madani, Mikkel Thorup, and Uri Zwick. Discounted deterministic markov decision processes and discounted all-pairs shortest paths. *ACM Transactions on Algorithms*, 6(2), 2010.

11 M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:269–311, 1997.

12 U. Zwick and M.S. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.

Full Abstraction for Resource Calculus with Tests*

Antonio Bucciarelli[1], Alberto Carraro[1,3], Thomas Ehrhard[1], and Giulio Manzonetto[2]

1 **Laboratoire PPS, CNRS, Université Paris-Diderot, Paris, France**
{antonio.bucciarelli,alberto.carraro,thomas.ehrhard}@pps.jussieu.fr
2 **Intelligent Systems, Radboud University, Nijmegen, The Netherlands**
g.manzonetto@cs.ru.nl
3 **Department of Computer Science, Ca'Foscari University, Venice, Italy**

Abstract

We study the semantics of a resource sensitive extension of the λ-calculus in a canonical reflexive object of a category of sets and relations, a relational version of the original Scott $\mathcal{D}_\infty$ model of the pure λ-calculus. This calculus is related to Boudol's resource calculus and is derived from Ehrhard and Regnier's differential extension of Linear Logic and of the λ-calculus. We extend it with new constructions, to be understood as implementing a very simple exception mechanism, and with a "must" parallel composition. These new operations allow to associate a context of this calculus with any point of the model and to prove full abstraction for the finite sub-calculus where ordinary λ-calculus application is not allowed. The result is then extended to the full calculus by means of a Taylor Expansion formula.

1998 ACM Subject Classification F.4.1 Lambda calculus and related systems

Keywords and phrases Resource lambda calculus, relational semantics, full abstraction, differential linear logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.97

1 Introduction

In concurrent calculi like CCS [11], guarded processes are resources which can be used only once by other processes. This fundamental linearity of resources leads naturally to non-determinism, since several agents (senders and receivers) can interact on the same channel. In general, various synchronization scenarios are possible, giving rise to different behaviours. On the other hand in the λ-calculus, a function (receiver) can duplicate its argument (sender) arbitrarily. Thanks to this asymmetry, the λ-calculus enjoys a strong determinism (Church-Rosser), but it lacks any form of control on resource handling.

Resource Lambda Calculi. Resource λ-calculi stem from an attempt to combine the functionality of the λ-calculus and the resource sensitivity of process calculi. Boudol has been the first to design a resource conscious functional programming language, the *resource λ-calculus*, extending the usual one along two directions [2]: a function is not necessarily applied to a single argument but can also be applied to a multiset of arguments called *resources*; a resource can be either linear (it must be used exactly once) or reusable (it can be used *ad libitum*). In this context, the evaluation of a function applied to a multiset of resources gives rise to several possible choices, corresponding to the different possibilities of

* This work was partially supported by NWO Project 612.000.936 CALMOC (CAtegorical and ALgebraic Models of Computation) and ANR Project ANR-07-BLAN-0324 CHOCO.

distributing the resources in the multiset among the occurrences of the formal parameter. From the viewpoint of concurrent programming, this was a natural step to take since one of the main features of this programming setting is the consumption of resources which cannot be copied. Milner's π-calculus [12] features this phenomenon in great generality, and Boudol's calculus keeps track of it in a functional setting.

Together with Regnier, Ehrhard observed that this idea of resource consumption can be understood as resulting from a *differential* extension of λ-calculus (and of Linear Logic) [6]. Instead of considering two kinds of resources, they defined two kinds of applications: the *ordinary* application and a *linear* one. In a simply typed setting, linear application of a term $M : A \to B$ to a multiset made of n terms $N_1, \ldots, N_n : A$, combined with ordinary application to a term $N : A$, corresponds to computing $M^{(n)}(N)(N_1, \ldots, N_n) : B$, where $M^{(n)}$ is the n-th derivative of M, which is of type $A \to (A^n \to B)$ and associates a symmetric n-linear map with any element of A. The symmetry of this multilinear map corresponds to the Schwarz Lemma of differential calculus and is implemented in the resource λ-calculus by the use of multisets for representing linear applications.

The main difference between the resource λ-calculus and the differential λ-calculus is that the first is lazy and is endowed with an explicit substitution mechanism. Therefore, Boudol's calculus is not an extension of the ordinary λ-calculus. Also, the resource λ-calculus is rather affine than linear, since depletable resources cannot be duplicated but can be erased. Another difference lies in the respective origins of these calculi: the resource λ-calculus originates from syntactical considerations related to the theory of concurrent processes, while the differential one arises from denotational models of linear logic where the existence of differential operations has been observed. These models are based on the well known relational model of Linear Logic and the interpretation of the new differential constructions is as natural and simple as the interpretation of the ordinary LL constructions.

Two main syntaxes have been proposed for the differential λ-calculus: Ehrhard and Regnier's original one [6], simplified by Vaux in [16], and Tranquilli's *resource calculus* of [15] whose syntax is close to Boudol's one. These calculi share a common semantical backbone as well as similar connections with differential Linear Logic and proof nets. We adopt roughly Tranquilli's syntax and call our calculus *$\partial\lambda$-calculus*.

Full Abstraction. A natural open problem when a new calculus is introduced is to characterize when two programs are operationally equivalent, namely when one can be replaced by the other in every context without noticing any difference with respect to a given observational equivalence. In this paper we prove a full abstraction result (a semantical characterization of operational equivalence) for the $\partial\lambda$-calculus in the spirit of [3]. As in that paper, we extend the language with a convergence testing mechanism. Implicitly, this extension already appears in [5], in a differential LL setting: it corresponds to the 0-ary tensor and par cells. To implement the corresponding extension of the λ-calculus, we introduce two sorts of expressions: the *terms* (variable, application, abstraction, "throw" $\bar{\tau}(P)$ where P is a test) and the *tests* (empty test, parallel composition of tests and "catch" $\tau(M)$ where M is a term). Parallel composition allows to combine tests in such a way that the combination succeeds if and only if each test succeeds. Outcomes of tests (convergence or divergence) are the only observations allowed in our calculus, and the corresponding contextual equivalence and preorder on terms constitute our main object of study.

This extended $\partial\lambda$-calculus, that we call *$\partial\lambda$-calculus with tests*, has a natural denotational interpretation in a model of the pure λ-calculus introduced by Bucciarelli, Ehrhard and Manzonetto in [4], which is indeed a denotational model of the differential pure nets of [5] as one can check easily. This model is a reflexive object $\mathcal{D}$ in the Kleisli category of the LL

model of sets and relations where $!X$ is the set of all finite multisets over X. An element of $\mathcal{D}$ can be described as a finite tree which alternates two kinds of layers: *multiplicative layers* where subtrees are indexed by natural numbers and *exponential layers* where subtrees are organized as non-empty multisets. To be more precise, $?\mathfrak{P}-?$ (negative) pairs of layers alternate with $\otimes-!$ (positive) pairs, respecting a strict polarity discipline very much in the spirit of Ludics [9]. The empty positive multiplicative tree corresponds to the empty tensor cell and the negative one to the empty par cell. The corresponding constructions τ, $\bar{\tau}$ are therefore quite easy to interpret.

We use this logical interpretation to turn the elements of $\mathcal{D}$ into $\partial\lambda$-calculus terms with tests. More precisely, with each element α of $\mathcal{D}$, we associate a test $\alpha^+(\!|\cdot|\!)$ with a hole $(\!|\cdot|\!)$ for a term, and we show that α belongs to the interpretation of a (closed) term M iff the test $\alpha^+(\!|M|\!)$ converges. From this fact, we derive a full abstraction result for the fragment of the $\partial\lambda$-calculus with tests in which all ordinary applications are trivial, that we call $\partial_0\lambda$-calculus with tests. To extend this result to the $\partial\lambda$-calculus with tests, we use the Taylor formula introduced in [6] which allows to turn any ordinary application into a sum of infinitely many linear applications of all possible arities. One exploits then the fact that the Taylor formula holds in the model, as well as a simulation lemma which relates the head reduction of a term with the head reduction of its Taylor expansion.

Contributions. The definability of the elements of $\mathcal{D}$ in the $\partial\lambda$-calculus with tests is the main conceptual contribution of this paper: it shows that, in the $\partial\lambda$-calculus with tests, the standard syntax versus semantics dichotomy is essentially meaningless. We also consider the use of the Taylor expansion to reduce the full abstraction problem to its $\partial_0\lambda$ version as an original and promising reduction technique. Notice that the tests added to the calculus are needed to develop this new methodology, although we conjecture they do not add discriminating power to the calculus (contrary to [3]).

Notations and basic definitions. We denote by $\mathbf{N}$ the set of natural numbers and by $\mathbf{1}$ an arbitrary singleton set. We write $\mathfrak{S}_k$ for *the set of all permutations of* $\{1,\dots,k\}$.

Let S be a set. We write $\mathcal{P}(S)$ (resp. $\mathcal{P}_{\mathrm{f}}(S)$) for the set of all (resp. finite) subsets of S. A *multiset* a over S is defined as an unordered list $a=[\alpha_1,\alpha_2,\dots]$ with repetitions such that $\alpha_i\in S$ for all indices i. A multiset a is called *finite* if it is a finite list, we denote by $\#a$ its cardinality. We write $\mathcal{M}_{\mathrm{f}}(S)$ for the set of all finite multisets over S. Given two multisets a,b we denote their *union* by $a\uplus b$. Given two finite sequences of multisets $\vec{a},\vec{b}$ of the same length n we define $\vec{a}\uplus\vec{b}=(a_1\uplus b_1,\dots,a_n\uplus b_n)$.

An operator $F(-)$ is *extended by linearity* by setting $F(\Sigma_i x_i)=\Sigma_i F(x_i)$.

2 The $\partial_0\lambda$-Calculus with Tests

We now introduce the *$\partial_0\lambda$-calculus with tests* which is the promotion-free fragment of the $\partial\lambda$-calculus with tests presented in Section 5. The $\partial_0\lambda$-calculus with tests has four syntactic categories: *terms* that are in functional position, *bags* that are in argument position and represent multisets of linear resources, *tests* that are "corked" multisets of terms having only two possible outcomes and *finite formal sums* representing all possible results of a computation. Formally, we have the following grammar:

$$(\Lambda^{\bar{\tau}})\ M,N,L,H ::= x \mid \lambda x.M \mid MP \mid \bar{\tau}(Q) \qquad \text{terms}$$
$$(\Lambda^{b})\ P ::= [L_1,\dots,L_k] \qquad \text{bags}$$
$$(\Lambda^{\tau})\ Q,R ::= \tau[L_1,\dots,L_k] \qquad \text{tests}$$
$$(\Lambda^{e})\ A,B ::= M \mid P \mid Q \qquad \text{expressions}$$

Tests are multisets of terms, the "τ" being a tag for distinguishing them from bags.

Throughout the paper, we will enforce the distinction between bags and tests by using systematically the following notational conventions.

- For bags, we use the usual multiset notation: $[]$ is the empty bag and $P \uplus P'$ is the union.
- For tests, ε is the empty multiset and $Q|R$ is the multiset union of Q and R. In other words, $\varepsilon = \tau[]$ and $\tau[L_1, \ldots, L_k] \mid \tau[L_{k+1}, \ldots, L_n] = \tau[L_1, \ldots, L_n]$.

Terms are the real protagonists of the $\partial_0\lambda$-calculus with tests. The term $\lambda x.M$ represents the λ-*abstraction* and MP the *application* of a term M to a bag P of linear resources. Thus, in $(\lambda x.M)P$, each resource in P is available exactly once for $\lambda x.M$ and if the number of occurrences of x in M "disagrees" with the cardinality of P then the result is 0 (see later, when sums are introduced). We set $\mathbf{I} := \lambda x.x$, where ':=' denotes definitional equality.

Tests are expressions which can produce two results: either *success*, represented by ε, or *failure*, represented by 0. The test $Q|R$ represents the (must-)parallel composition of Q and R (i.e., $Q|R$ succeeds if both Q and R succeed). The composition is parallel in the sense that the order of evaluation is inessential.

The operator $\bar{\tau}(\cdot)$ allows to build a term out of a test: intuitively, the term $\bar{\tau}(Q)$ may be thought of as Q preceded by an infinite sequence of dummy λ-abstractions. Dually, the "cork construction" $\tau[L_1, \ldots, L_k]$ may be thought of as an operator applying to all its arguments an infinite sequence of empty bags. This suggests that it is sound to reduce $\tau[\bar{\tau}(Q)]$ to Q.

Hence the term $\bar{\tau}(Q)$ raises an exception encapsulating Q and the test $\tau[L_1, \ldots, L_k]$ catches the exception possibly raised by any of the L_i's and replaces L_i by the multiset of terms encapsulated in that exception. The context of the exception is thrown away by the dummy abstractions of $\bar{\tau}$ and the dummy applications of τ. A test needs to catch an exception in order to succeed; for instance, $\tau[M]$ fails as soon as M is a $\bar{\tau}$-free, closed term.

We will write $\|_{i=1}^{n} R_i$ for $R_1|\cdots|R_n$; obviously we have $\|_{i=1}^{0} R_i = \varepsilon$ and $\|_{i=1}^{1} R_i = R_1$.

Expressions are either terms, bags or tests.

Sums. Let $\mathbf{2}$ be the semiring $\{0, 1\}$ with $1 + 1 = 1$ and multiplication defined in the obvious way. For any set A, we write $\mathbf{2}\langle A\rangle$ for the free $\mathbf{2}$-module generated by A, so that $\mathbf{2}\langle A\rangle \cong \mathcal{P}_{\mathrm{f}}(A)$ with addition corresponding to union, and scalar multiplication defined in the obvious way. However we prefer to keep the algebraic notations for elements of $\mathbf{2}\langle A\rangle$, hence set unions will be denoted by $+$ and the empty set by 0. This amounts to say that $\mathbf{2}\langle\Lambda^{\bar{\tau}}\rangle$ (resp. $\mathbf{2}\langle\Lambda^{\tau}\rangle$, $\mathbf{2}\langle\Lambda^{b}\rangle$) is the set of finite formal sums of terms (resp. tests, bags) with an idempotent sum. We also set $\mathbf{2}\langle\Lambda^{e}\rangle = \mathbf{2}\langle\Lambda^{\tau}\rangle \cup \mathbf{2}\langle\Lambda^{\bar{\tau}}\rangle \cup \mathbf{2}\langle\Lambda^{b}\rangle$. This is an abuse of notation as $\mathbf{2}\langle\Lambda^{e}\rangle$ here does not denote the $\mathbf{2}$-module generated over $\Lambda^{\tau} \cup \Lambda^{\bar{\tau}} \cup \Lambda^{b}$, but rather the union of the three $\mathbf{2}$-modules; this means that sums should be taken only in the same sort.

Typical metavariables to denote sums are: $\mathbb{M}, \mathbb{N}, \mathbb{L}, \mathbb{H} \in \mathbf{2}\langle\Lambda^{\bar{\tau}}\rangle$, $\mathbb{P} \in \mathbf{2}\langle\Lambda^{b}\rangle$, $\mathbb{Q}, \mathbb{R} \in \mathbf{2}\langle\Lambda^{\tau}\rangle$, $\mathbb{A}, \mathbb{B} \in \mathbf{2}\langle\Lambda^{e}\rangle$. The α-*equivalence* relation and the set $\mathrm{FV}(\mathbb{A})$ of *free variables of* $\mathbb{A}$ are defined as usual, like in the ordinary λ-calculus [1]. We write $\deg_x(\mathbb{A})$ for the number of free occurrences of x in $\mathbb{A}$. Hereafter, (sums of) expressions are considered up to α-equivalence.

2.1 Two Kinds of Substitutions

Notice that the grammar for terms and tests does not include any sums, so they may arise only on the "surface". However, as syntactic sugar – and *not* as actual syntax – we extend all the constructors to sums by multilinearity, setting for instance $(\Sigma_i M_i)(\Sigma_j P_j) := \Sigma_{i,j} M_i P_j$, in such a way that the following equations hold:

$$\lambda x.(\Sigma_i M_i) = \Sigma_i \lambda x.M_i \qquad \mathbb{M}(\Sigma_i P) = \Sigma_i \mathbb{M} P_i \qquad (\Sigma_i M_i)\mathbb{P} = \Sigma_i M_i \mathbb{P} \qquad \tau[\Sigma_i M_i] = \Sigma_i \tau[M_i]$$
$$(\Sigma_i R_i) \mid \mathbb{Q} = \Sigma_i\ R_i \mid \mathbb{Q} \qquad [\Sigma_i L_i] = \Sigma_i [L_i] \qquad (\Sigma_i P_i) \uplus \mathbb{P} = \Sigma_i P_i \uplus \mathbb{P} \qquad \bar{\tau}(\Sigma_i R_i) = \Sigma_i \bar{\tau}(R_i)$$

As an example of this *extended (meta-)syntax*, we may write $(x_1 + x_2)[y_1 + y_2]$ for $x_1[y_1] + x_1[y_2] + x_2[y_1] + x_2[y_2]$. This kind of meta-syntactic notation is discussed thoroughly in [8].

Observe that in the particular case of empty sums, we get $\lambda x.0 := 0$, $M0 := 0$, $0P := 0$, $\tau[0] := 0$, $\bar{\tau}(0) := 0$, $R|0 := 0$, $[0] := 0$, $0 \uplus P := 0$. Thus 0 annihilates any term, bag or test.

We now introduce two kinds of substitutions: the usual λ-calculus substitution and a linear one, which is proper to differential and resource calculi (see [2, 6, 15]).

Let $A \in \Lambda^e$ and $N \in \Lambda^{\bar{\tau}}$. The *(capture-free) substitution of N for x in A*, denoted by $A\{N/x\}$, is defined as usual. Accordingly, $A\{\mathbb{N}/x\}$ denotes a term of the extended syntax. Finally, we extend this operation to sums as in $\mathbb{A}\{\mathbb{N}/x\}$ by linearity in $\mathbb{A}$.

The *linear (capture-free) substitution of N for x in A*, denoted by $A\langle N/x\rangle$, is defined as follows (in this definition we strongly use the extended syntax):

$$y\langle N/x\rangle = \begin{cases} N & \text{if } y = x, \\ 0 & \text{otherwise,} \end{cases} \qquad [L_1, \dots, L_k]\langle N/x\rangle = \Sigma_{i=1}^{k}[L_1, \dots, L_i\langle N/x\rangle \dots, L_k],$$

$$\tau[L_1, \dots, L_k]\langle N/x\rangle = \Sigma_{i=1}^{k}\tau[L_1, \dots, L_i\langle N/x\rangle, \dots, L_k],$$

$$(MP)\langle N/x\rangle = M\langle N/x\rangle P + M(P\langle N/x\rangle), \qquad \bar{\tau}(Q)\langle N/x\rangle = \bar{\tau}(Q\langle N/x\rangle),$$

$$(\lambda y.M)\langle N/x\rangle = \lambda y.M\langle N/x\rangle, \qquad \text{(in the abstraction case we assume wlog } x \neq y).$$

Roughly speaking, linear substitution replaces the resource to *exactly one* linear free occurrence of x. If there is no occurrence of x then the result is 0. In presence of multiple occurrences, all possible choices are made and the result is the sum of them. For example, we have $(y[x][x])\langle \mathbf{I}/x\rangle = y[\mathbf{I}][x] + y[x][\mathbf{I}]$.

An example of regular substitution is $(x[x])\{(z_1 + z_2)/x\} = z_1[z_1] + z_1[z_2] + z_2[z_1] + z_2[z_2]$.

Turning to the extension of linear substitution to sums: the term $A\langle \mathbb{N}/x\rangle$ belongs to the extended syntax, and we extend it to sums as in $\mathbb{A}\langle \mathbb{N}/x\rangle$ by linearity in $\mathbb{A}$, as we did for usual substitution.

Observe that $\mathbb{A}\langle \mathbb{N}/x\rangle$ is linear in $\mathbb{A}$ and in $\mathbb{N}$, whereas $\mathbb{A}\{\mathbb{N}/x\}$ is linear in $\mathbb{A}$ but not in $\mathbb{N}$.

Linear substitutions commute in the sense expressed by the next lemma.

▶ **Lemma 1** (Schwarz Lemma, cf. [6]). *For $\mathbb{A} \in \mathbf{2}\langle \Lambda^e\rangle$, $\mathbb{M}, \mathbb{N} \in \mathbf{2}\langle \Lambda^{\bar{\tau}}\rangle$ and $y \notin \mathrm{FV}(\mathbb{M}) \cup \mathrm{FV}(\mathbb{N})$ we have $\mathbb{A}\langle \mathbb{M}/y\rangle\langle \mathbb{N}/x\rangle = \mathbb{A}\langle \mathbb{N}/x\rangle\langle \mathbb{M}/y\rangle + \mathbb{A}\langle \mathbb{M}\langle \mathbb{N}/x\rangle/y\rangle$. In particular, if $x \notin \mathrm{FV}(\mathbb{M})$ then the two substitutions commute.*

Given a bag $P = [L_1, \dots, L_k]$ such that $x \notin \mathrm{FV}(P)$ it makes sense to define $\mathbb{A}\langle P/x\rangle := \mathbb{A}\langle L_1/x\rangle \cdots \langle L_k/x\rangle$, because this expression does not depend on the enumeration $L_1, \dots, L_k$. In particular, $\mathbb{A}\langle []/x\rangle = \mathbb{A}$. Given bags $P_1, \dots, P_n$ we set $\mathbb{A}\langle \vec{P}/\vec{x}\rangle := \mathbb{A}\langle P_1/x_1\rangle \cdots \langle P_n/x_n\rangle$.

2.2 The Operational Semantics

We are going to introduce the reduction rules defining the operational semantics of the $\partial_0\lambda$-calculus with tests and show that it enjoys Church-Rosser and strong normalization, even in the untyped version of the calculus.

▶ **Definition 2.** The reduction semantics of the $\partial_0\lambda$-calculus with tests is generated by the following rules (in the abstraction case we suppose wlog that $x \notin \mathrm{FV}(P)$):

$$(\lambda x.M)P \to_\beta M\langle P/x\rangle\{0/x\}, \qquad \bar{\tau}(Q)P \to_{\bar{\tau}} \begin{cases} \bar{\tau}(Q) & \text{if } P = [], \\ 0 & \text{otherwise,} \end{cases}$$

$$\tau[\lambda x.M]|R \to_\tau \tau[M\{0/x\}]|R, \qquad \tau[\bar{\tau}(Q)]|R \to_\gamma Q|R.$$

Notice that the reduction preserves the sort of an expression in the sense that terms rewrite to (sums of) terms and tests to (sums of) tests. Also remark that, if M has k free occurrences of x (represented by $x^1, \dots, x^k$) then we have $M\langle L_1/x\rangle \cdots \langle L_k/x\rangle\{0/x\} = \Sigma_{\sigma \in \mathfrak{S}_k} M\{L_{\sigma(1)}/x^1, \dots, L_{\sigma(k)}/x^k\}$; it is equal to 0 otherwise (namely, when $\deg_x(M) \neq k$).

We denote by $\to \subseteq \mathbf{2}\langle\Lambda^e\rangle \times \mathbf{2}\langle\Lambda^e\rangle$ the contextual closure of $\to_\beta \cup \to_{\bar\tau} \cup \to_\tau \cup \to_\gamma$. In particular, parallel composition is treated asynchronously, thus $R \to \mathbb{R}$ entails $Q|R \to Q|\mathbb{R}$ (which is equal to $\mathbb{R}|Q$). This means, for instance, that if $L \to \bar\tau(Q)$, then $\tau[L, \vec{N}] \to \tau[\bar\tau(Q), \vec{N}] \to Q \mid \tau[\vec{N}]$. We write $\twoheadrightarrow$ for the transitive and reflexive closure of $\to$.

▶ **Definition 3.** An expression A is *in normal form* (*nf*, for short) if there is no $\mathbb{B}$ such that $A \to \mathbb{B}$. A sum of expressions $\mathbb{A}$ *is in nf* if $\mathbb{A} \neq 0$ and all its summands are in nf.

It is easy to check that a term $M \in \Lambda^{\bar\tau}$ is in normal-form if either $M = \lambda\vec{x}.yP_1 \cdots P_n$ or $M = \lambda\vec{x}.\bar\tau(\|_{i=1}^n \tau[y_i P_1^i \cdots P_{k_i}^i])$ where $n \geq 0$, $k_i \geq 0$ and each P_j^i is a bag of terms in nf.

▶ **Theorem 4.** *The $\partial_0\lambda$-calculus with tests is strongly normalizing and Church-Rosser.*

Proof. The fact that there are no infinite reduction chains is trivial, since every reduction step decreases the size of an expression (which is straightforward to define). For the Church-Rosser property just check local confluence and conclude by Newman's lemma. ◀

▶ **Lemma 5.** *For any closed term M, either $\tau[M] \twoheadrightarrow \varepsilon$ or $\tau[M] \twoheadrightarrow 0$.*

Proof. As $\partial_0\lambda$-calculus with tests is strongly normalizing, we have that $M \twoheadrightarrow \Sigma_{i=1}^k M_i$, where each M_i is a closed nf. If $k = 0$ then $\tau[M] \twoheadrightarrow 0$ since $\tau[0] = 0$. Otherwise for each M_i there are two possibilities:

- $M_i = \lambda\vec{x}.x_j P_1 \cdots P_n$ with $x_j \in \vec{x}$ and $n \geq 0$. Then $\tau[M_i] \twoheadrightarrow \tau[(x_j P_1 \cdots P_n)\{0/\vec{x}\}] = \tau[0] = 0$.
- $M_i = \lambda\vec{x}.\bar\tau(\|_{j=1}^n \tau[x_j P_1^j \cdots P_{k_j}^j])$ with $n \geq 0$ and $x_j \in \vec{x}$. If $n = 0$ then we have $\|_{j=1}^n \tau[x_j P_1^j \cdots P_{k_j}^j] = \varepsilon$ and $\tau[\lambda\vec{x}.\bar\tau(\varepsilon)] \twoheadrightarrow \tau[\bar\tau(\varepsilon)] \to \varepsilon$. If $n > 0$, then we have $\tau[M_i] \twoheadrightarrow \tau[\bar\tau(\|_{j=1}^n \tau[0 P_1^j\{0/\vec{x}\} \cdots P_{k_j}^j\{0/\vec{x}\}])] = 0$.

We conclude since $\tau[M] \twoheadrightarrow \Sigma_{i=1}^k \tau[M_i]$, and this latter expression reduces to a finite (possibly empty) sum of ε's, which is thus equal either to 0 or to ε. ◀

▶ **Corollary 6.** *If R is a closed test then either $R \twoheadrightarrow \varepsilon$ or $R \twoheadrightarrow 0$.*

Contexts. A *test-context* $C(\!|\cdot|\!)$ is a test having one occurrence of a *hole*, denoted by $(\!|\cdot|\!)$, appearing in term-position. The set of test-contexts is denoted by $\Lambda^\tau_{(\!|\cdot|\!)}$. Given $M \in \Lambda^{\bar\tau}$ we indicate by $C(\!|M|\!)$ the test resulting by blindly replacing M for the hole (allowing capture of free variables) in $C(\!|\cdot|\!)$. We say that $C(\!|\cdot|\!)$ is *closed* if it contains no free variable; it is *closing* M if $C(\!|M|\!)$ is closed. We say that a test Q *converges*, and we write $Q{\downarrow}$, if $Q \twoheadrightarrow \varepsilon$.

▶ **Definition 7.** The *operational pre-order* $\sqsubseteq_{\mathcal{O}}$ is defined by:

$$M \sqsubseteq_{\mathcal{O}} N \Leftrightarrow \forall C(\!|\cdot|\!) \in \Lambda^\tau_{(\!|\cdot|\!)} \text{ closing } M, N \ (C(\!|M|\!){\downarrow} \Rightarrow C(\!|N|\!){\downarrow}).$$

We set $M \approx_{\mathcal{O}} N$ if and only if $M \sqsubseteq_{\mathcal{O}} N$ and $N \sqsubseteq_{\mathcal{O}} M$.

The choice of test-convergence as the basic observation in our calculus is very natural. Indeed, tests provide a canonical notion of observation since – by design – they either converge (to ε) or diverge.

3 A Relational Semantics

This section is devoted to build a relational model $\mathcal{D}$ of $\partial_0\lambda$-calculus with tests, that has been first introduced in [4] as a model of the ordinary λ-calculus. We first give a sketchy presentation of the Cartesian closed category **MRel** where $\mathcal{D}$ lives.

The objects of **MRel** are all the sets. A morphism from S to T is a relation from $\mathcal{M}_{\mathrm{f}}(S)$ to T, in other words, $\mathbf{MRel}(S,T) = \mathcal{P}(\mathcal{M}_{\mathrm{f}}(S) \times T)$. The identity of S is the relation $\mathrm{Id}_S = \{([\alpha], \alpha) : \alpha \in S\}$. The composition of $s : S \to T$ and $t : T \to U$ is defined by:

$$t \circ s = \{(m, c) : \exists (m_1, \beta_1), ..., (m_k, \beta_k) \in s \text{ such that } m = \uplus_{i=1}^{k} m_i \text{ and } ([\beta_1, ..., \beta_k], c) \in t\}.$$

The categorical product $S \,\&\, T$ of two sets S and T is their disjoint union. The terminal object is the empty set $\emptyset$. The exponential object internalizing $\mathbf{MRel}(S,T)$ is $\mathcal{M}_{\mathrm{f}}(S) \times T$.

An infinite sequence $\alpha = (a_1, a_2, \dots)$ of multisets is *quasi-finite* if $a_i = []$ holds for all but a finite number of indices i. If S is a set, we denote by $\mathcal{M}_{\mathrm{f}}(S)^{(\omega)}$ the set of all quasi-finite $\mathbf{N}$-indexed sequences of multisets over S.

We build a family of sets $(D_n)_{n \in \mathbf{N}}$ as follows: $D_0 = \emptyset$, $D_{n+1} = \mathcal{M}_{\mathrm{f}}(D_n)^{(\omega)}$. Since the operation $S \mapsto \mathcal{M}_{\mathrm{f}}(S)^{(\omega)}$ is monotonic with respect to inclusion and $D_0 \subseteq D_1$, we have $D_n \subseteq D_{n+1}$ for all $n \in \mathbf{N}$. Finally, we set $\mathcal{D} = \bigcup_{n \in \mathbf{N}} D_n$.

To define an isomorphism between $\mathcal{D}$ and $\mathcal{M}_{\mathrm{f}}(\mathcal{D}) \times \mathcal{D}$ just remark that every element $\alpha = (a_1, a_2, a_3, \dots) \in \mathcal{D}$ stands for the pair $(a_1, (a_2, a_3, \dots))$ and *vice versa*. Hence $\mathcal{D} \cong [\mathcal{D} \Rightarrow \mathcal{D}]$ (we have a canonical bijection between these two sets, and therefore an isomorphism in **MRel**). Given $\alpha = (a_1, a_2, a_3, \dots) \in \mathcal{D}$ and $a \in \mathcal{M}_{\mathrm{f}}(\mathcal{D})$, we write $a :: \alpha$ for the element $(a, a_1, a_2, a_3, \dots) \in \mathcal{D}$. We set $* = ([], [], \dots, [], \dots) \in \mathcal{D}$. Remark that $[] :: * = *$.

3.1 Interpreting the $\partial_0\lambda$-calculus with tests

For all terms M, bags P, tests Q and repetition-free sequences $\vec{x}, \vec{y}, \vec{z}$ respectively containing the free variables of M, P, Q, we define by mutual induction the interpretations $[\![M]\!]_{\vec{x}} : \mathcal{D}^n \to \mathcal{D}$, $[\![P]\!]_{\vec{y}} : \mathcal{D}^m \to \mathcal{M}_{\mathrm{f}}(\mathcal{D})$ and $[\![Q]\!]_{\vec{z}} : \mathcal{D}^k \to \mathbf{1}$ (n, m, k are the lengths of $\vec{x}, \vec{y}, \vec{z}$) as follows[1]:

- $[\![x_i]\!]_{\vec{x}} = \{(([], \dots, [], [\alpha], [], \dots, []), \alpha) : \alpha \in \mathcal{D}\}$, where $[\alpha]$ stands in i-th position,
- $[\![\lambda y.M]\!]_{\vec{x}} = \{(\vec{a}, b :: \alpha) : ((\vec{a}, b), \alpha) \in [\![M]\!]_{\vec{x},y}\}$, where we suppose wlog that $y \notin \vec{x}$,
- $[\![MP]\!]_{\vec{x}} = \{(\vec{a}_0 \uplus \vec{a}_1, \alpha) : \exists b \in \mathcal{M}_{\mathrm{f}}(\mathcal{D})\ (\vec{a}_0, b :: \alpha) \in [\![M]\!]_{\vec{x}},\ (\vec{a}_1, b) \in [\![P]\!]_{\vec{x}}\}$,
- $[\![[L_1, \dots, L_k]]\!]_{\vec{x}} = \{(\uplus_{i=1}^{k} \vec{a}_i, [\beta_1, \dots, \beta_k]) : (\vec{a}_i, \beta_i) \in [\![L_i]\!]_{\vec{x}},\ 1 \leq i \leq k\}$,
- $[\![\bar{\tau}(Q)]\!]_{\vec{x}} = \{(\vec{a}, *) : \vec{a} \in [\![Q]\!]_{\vec{x}}\}$,
- $[\![\tau[M]]\!]_{\vec{x}} = \{\vec{a} : (\vec{a}, *) \in [\![M]\!]_{\vec{x}}\}$,
- $[\![Q|R]\!]_{\vec{x}} = \{\vec{a}_0 \uplus \vec{a}_1 : \vec{a}_0 \in [\![Q]\!]_{\vec{x}}, \vec{a}_1 \in [\![R]\!]_{\vec{x}}\}$,
- $[\![\varepsilon]\!]_{\vec{x}} = \{([], \dots, [])\}$.

The interpretation is then extended to sums by setting $[\![\Sigma_{i=1}^{k} A_i]\!]_{\vec{x}} = \cup_{i=1}^{k} [\![A_i]\!]_{\vec{x}}$. Note that $[\![[]]\!]_{\vec{x}} = \{([], \dots, [])\} \in \mathcal{M}_{\mathrm{f}}(\mathcal{D})^{n+1}$. Since every test R is of the form $\tau[L_1, \dots, L_k]$ we might define its interpretation directly as $[\![R]\!]_{\vec{x}} = \{\uplus_{i=1}^{k} \vec{a}_i : (\vec{a}_i, *) \in [\![L_i]\!]_{\vec{x}}, 1 \leq i \leq k\}$.

Hereafter, whenever we write $[\![A]\!]_{\vec{x}}$ we suppose that $\vec{x}$ is a repetition-free list of variables of length n containing $\mathrm{FV}(A)$. Moreover, we will sometimes silently use the fact $[\![M]\!]_{\vec{x},y} = \{((\vec{a}, []), \alpha) : (\vec{a}, \alpha) \in [\![M]\!]_{\vec{x}}\}$ whenever $y \notin \vec{x}$.

Clearly the interpretation is monotonic, i.e., for any test context $C(\!|\cdot|\!)$ with free variables $\vec{y}$, if $[\![M]\!]_{\vec{x}} \subseteq [\![N]\!]_{\vec{x}}$ then $[\![C(\!|M|\!)]\!]_{\vec{x},\vec{y}} \subseteq [\![C(\!|N|\!)]\!]_{\vec{x},\vec{y}}$.

The following substitution lemmas are needed for proving the invariance of the interpretation under reduction. The proofs are lengthy but not difficult, and are omitted.

[1] Since $\mathcal{M}_{\mathrm{f}}(S \,\&\, T) \cong \mathcal{M}_{\mathrm{f}}(S) \times \mathcal{M}_{\mathrm{f}}(T)$ we have, up to isomorphism, $[\![M]\!]_{\vec{x}} \subseteq \mathcal{M}_{\mathrm{f}}(\mathcal{D})^n \times \mathcal{D}$, $[\![P]\!]_{\vec{y}} \subseteq \mathcal{M}_{\mathrm{f}}(\mathcal{D})^{m+1}$ and $[\![Q]\!]_{\vec{z}} \subseteq \mathcal{M}_{\mathrm{f}}(\mathcal{D})^k \times \mathbf{1} \cong \mathcal{M}_{\mathrm{f}}(\mathcal{D})^k$.

▶ **Lemma 8** (Linear Substitution Lemma). *Let $M \in \Lambda^{\bar\tau}, Q \in \Lambda^{\tau}$ and $P = [L_1, \dots, L_k] \in \Lambda^b$ such that $\deg_y(M) = \deg_y(Q) = k$. We have:*

(i) $(\vec a, \alpha) \in [\![M\langle P/y\rangle]\!]_{\vec x}$ *iff there exist* $(\vec a_i, \beta_i) \in [\![L_i]\!]_{\vec x}$ *(for* $1 \le i \le k$*) and* $\vec a_0 \in \mathcal{M}_{\mathrm f}(\mathcal{D})^n$ *such that* $((\vec a_0, [\beta_1, \dots, \beta_k]), \alpha) \in [\![M]\!]_{\vec x, y}$ *and* $\uplus_{i=0}^k \vec a_i = \vec a$.

(ii) $\vec a \in [\![Q\langle P/y\rangle]\!]_{\vec x}$ *iff there exist* $(\vec a_i, \beta_i) \in [\![L_i]\!]_{\vec x}$ *(for* $1 \le i \le k$*) and* $\vec a_0 \in \mathcal{M}_{\mathrm f}(\mathcal{D})^n$ *such that* $(\vec a_0, [\beta_1, \dots, \beta_k]) \in [\![Q]\!]_{\vec x, y}$ *and* $\uplus_{i=0}^k \vec a_i = \vec a$.

▶ **Lemma 9** (Regular Substitution Lemma). *Let $M \in \Lambda^{\bar\tau}$, $Q \in \Lambda^{\tau}$ and $\mathbb{N} \in \mathbf{2}\langle\Lambda^{\bar\tau}\rangle$. We have:*

(i) $(\vec a, \alpha) \in [\![M\{\mathbb{N}/y\}]\!]_{\vec x}$ *iff* $\exists k \in \mathbf{N}$, $\exists \beta_1, \dots, \beta_k \in \mathcal{D}$, $\exists \vec a_0, \dots, \vec a_k \in \mathcal{M}_{\mathrm f}(\mathcal{D})^n$ *such that* $(\vec a_i, \beta_i) \in [\![\mathbb{N}]\!]_{\vec x}$ *(for* $1 \le i \le k$*),* $((\vec a_0, [\beta_1, \dots, \beta_k]), \alpha) \in [\![M]\!]_{\vec x, y}$ *and* $\vec a = \uplus_{j=0}^k \vec a_j$,

(ii) $\vec a \in [\![Q\{\mathbb{N}/y\}]\!]_{\vec x}$ *iff* $\exists k \in \mathbf{N}$, $\exists \beta_1, \dots, \beta_k \in \mathcal{D}$, $\exists \vec a_0, \dots, \vec a_k \in \mathcal{M}_{\mathrm f}(\mathcal{D})^n$ *such that* $(\vec a_i, \beta_i) \in [\![\mathbb{N}]\!]_{\vec x}$ *(for* $1 \le i \le k$*) and* $(\vec a_0, [\beta_1, \dots, \beta_k]) \in [\![Q]\!]_{\vec x, y}$ *and* $\vec a = \uplus_{j=0}^k \vec a_j$.

The substitution lemmas above generalize straightforwardly to sums. Although Lemma 9 is stated in full generality, for the $\partial_0\lambda$-calculus with tests it is only useful for $\mathbb{N} = 0$. However, this formulation will be needed in Section 5 for the $\partial\lambda$-calculus with tests.

▶ **Theorem 10.** *$\mathcal{D}$ is a model of the $\partial_0\lambda$-calculus with tests, i.e., if $\mathbb{A} \twoheadrightarrow \mathbb{B}$ then $[\![\mathbb{A}]\!]_{\vec x} = [\![\mathbb{B}]\!]_{\vec x}$.*

Proof. It is easy to check that the interpretation is contextual. The fact that the semantics is invariant under reduction follows from Lemmas 8 and 9. ◀

4 First Full Abstraction Results

A model is *equationally fully abstract* (*FA*, for short) if the equivalence induced on terms by their interpretations is exactly $\approx_{\mathcal{O}}$; it is *inequationally FA* if the induced preorder is $\sqsubseteq_{\mathcal{O}}$. Every inequationally FA model is also FA. In this section we prove that $\mathcal{D}$ is inequationally FA for the $\partial_0\lambda$-calculus (Thm. 19), i.e., that $[\![M]\!]_{\vec x} \subseteq [\![N]\!]_{\vec x}$ iff $M \sqsubseteq_{\mathcal{O}} N$.

4.1 Building Separating Test-Contexts

In this section we are going to associate a test-context $\alpha^+(\!|\cdot|\!)$ with each element $\alpha \in \mathcal{D}$, the idea being that – for every closed term M – we have $\alpha \in [\![M]\!]$ iff $\alpha^+(\!|M|\!)$ converges.

▶ **Definition 11.** Let $\alpha \in \mathcal{D}$. The *rank of* α, written $\mathrm{rk}(\alpha)$, is the least $n \in \mathbf{N}$ such that $\alpha \in D_{n+1}$; the *length of* α, written $\ell(\alpha)$, is 0 if $\alpha = *$, and it is the unique r such that $\alpha = a_1 :: \cdots :: a_r :: *$ with $a_r \neq []$, otherwise.

Note that if $\alpha = a_1 :: \cdots :: a_r :: *$ then for all $1 \le i \le r$ and for all $\alpha' \in a_i$ we have $\mathrm{rk}(\alpha) > \mathrm{rk}(\alpha')$. Hence $\mathrm{rk}(\alpha) = 0$ entails $\alpha = *$ and the following definition is well-founded.

▶ **Definition 12.** For $\alpha \in \mathcal{D}$ of the form $\alpha = [\alpha_1^1, \dots, \alpha_{k_1}^1] :: \cdots :: [\alpha_1^r, \dots, \alpha_{k_r}^r] :: *$ with $\ell(\alpha) = r$, define by mutual induction a closed term α^- and a test-context $\alpha^+(\!|\cdot|\!)$ as follows:

- $\alpha^- = \lambda x_1 \dots x_r.\bar\tau(\|_{i=1}^r ((\alpha_1^i)^+(\!|x_i|\!) \,|\, \cdots \,|\, (\alpha_{k_i}^i)^+(\!|x_i|\!)))$,
- $\alpha^+(\!|\cdot|\!) = \tau[(\!|\cdot|\!)[(\alpha_1^1)^-, \dots, (\alpha_{k_1}^1)^-] \cdots [(\alpha_1^r)^-, \dots, (\alpha_{k_r}^r)^-]]$.

Given $a = [\alpha_1, \dots, \alpha_k]$ we set $a^- = [\alpha_1^-, \dots, \alpha_k^-]$.

For instance, we have $*^- = \bar\tau(\varepsilon)$ (as the empty parallel composition is equal to ε) and $*^+(\!|\cdot|\!) = \tau[(\!|\cdot|\!)]$.

The next lemma, along with its corollaries, shows the interplay between the elements of $\mathcal{D}$ and the terms/tests of Definition 12. It provides the main motivation for our extension of the $\partial\lambda$-calculus.

▶ **Lemma 13.** *Let $\alpha \in \mathcal{D}$. Then:*
(i) $[\![\alpha^-]\!] = \{\alpha\}$,
(ii) $[\![\alpha^+(\!|x|\!)]\!]_x = \{[\alpha]\}$.

Proof. The points (i) and (ii) are proved simultaneously by induction on $\mathrm{rk}(\alpha)$. We write IH(i) and IH(ii) for the induction hypotheses concerning (i) and (ii), respectively.

If $\mathrm{rk}(\alpha) = 0$ then $\alpha = *$, hence $[\![*^-]\!] = [\![\bar{\tau}(\varepsilon)]\!] = \{*\}$ and $[\![*^+(\!|x|\!)]\!]_x = [\![\tau[x]]\!]_x = \{[*]\}$.

If $\mathrm{rk}(\alpha) > 0$ and $\ell(\alpha) = r$, then we have $\alpha = a_1 :: \cdots :: a_r :: *$ with $a_i = [\alpha_1^i, \ldots, \alpha_{k_i}^i]$ for $1 \leq i \leq r$.

We prove (i). Remember that by definition $[\![\alpha^-]\!] = [\![\lambda y_1 \ldots y_r.\bar{\tau}(\|_{i=1}^r \|_{j=1}^{k_i} (\alpha_j^i)^+(\!|y_i|\!))]\!]$. So we have $\beta \in [\![\alpha^-]\!]$ iff $\beta = b_1 :: \cdots :: b_r :: *$ and for all $1 \leq i \leq r$, $1 \leq j \leq k_i$ there is $\vec{d}_j^{\,i} \in [\![(\alpha_j^i)^+(\!|y_i|\!)]\!]_{\vec{y}}$ such that $\vec{b} = \uplus_{i=1}^r \uplus_{j=1}^{k_i} \vec{d}_j^{\,i}$. By IH(ii) we have $\vec{d}_j^{\,i} \in [\![(\alpha_j^i)^+(\!|y_i|\!)]\!]_{\vec{y}}$ iff $\vec{d}_j^{\,i} = (\vec{[]}, [\alpha_j^i], \vec{[]})$ where $[\alpha_j^i]$ appears in i-th position. Therefore $\uplus_{j=1}^{k_i} \vec{d}_j^{\,i} = (\vec{[]}, a_i, \vec{[]})$ and $b_i = a_i$ for every index i. Thus $\beta = \alpha$.

We prove (ii). By def. $[\![\alpha^+(\!|x|\!)]\!]_x = [\![\tau[x a_1^- \cdots a_r^-]]\!]_x$. So we have $c \in [\![\alpha^+(\!|x|\!)]\!]_x$ iff there are $b_i = [\beta_i^i, \ldots, \beta_{k_i}^i]$, $c_0, c_1^i, \ldots, c_{k_i}^i \in \mathcal{M}_{\mathrm{f}}(\mathcal{D})$ (for $1 \leq i \leq r$) such that $(c_0, b_1 :: \cdots :: b_r :: *) \in [\![x]\!]_x$, $(c_j^i, \beta_j^i) \in [\![(\alpha_j^i)^-]\!]_x$ (for all $1 \leq i \leq r$ and $1 \leq j \leq k_i$) and $c = c_0 \uplus (\uplus_{i=1}^r \uplus_{j=1}^{k_i} c_j^i)$. As, by IH(i), $[\![(\alpha_{j_i}^i)^-]\!]_x = \{([], \alpha_j^i)\}$ we get $c_j^i = []$ and $\beta_j^i = \alpha_j^i$. Thus $c = c_0$, $\alpha = b_1 :: \cdots :: b_r :: *$ and from this it follows that $(c, \alpha) \in [\![x]\!]_x$. We conclude that $c = [\alpha]$. ◀

▶ **Corollary 14.** $[\![\alpha^+(\!|M|\!)]\!]_{\vec{x}} = \{\vec{c} : (\vec{c}, \alpha) \in [\![M]\!]_{\vec{x}}\}$.

Proof. By Lemma 13(ii) we have $[\![\alpha^+(\!|y|\!)]\!]_{\vec{x},y} = \{([], \ldots, [], [\alpha])\}$. As $\alpha^+(\!|\cdot|\!)$ does not have outer λ-abstractions we have $\alpha^+(\!|M|\!) = \alpha^+(\!|y|\!)\langle [M]/y \rangle$. We then apply Lemma 8 to conclude. ◀

▶ **Corollary 15.** *All finite subsets of $\mathcal{D}$ are definable.*

Proof. By Lemma 13(i), for every finite set $u = \{\alpha_1, \ldots, \alpha_k\}$ we have $[\![\alpha_1^- + \cdots + \alpha_k^-]\!] = u$. ◀

Lemma 13 reveals the behaviour of a test-context $\alpha^+(\!|\cdot|\!)$ when applied to a term β^-.

▶ **Corollary 16.** *Let $\alpha, \beta \in \mathcal{D}$. If $\alpha = \beta$ then $\alpha^+(\!|\beta^-|\!) \twoheadrightarrow \varepsilon$, otherwise $\alpha^+(\!|\beta^-|\!) \twoheadrightarrow 0$.*

Proof. By Lemma 13, $[\![\alpha^+(\!|\beta^-|\!)]\!] = \{()\} \subseteq \mathcal{M}_{\mathrm{f}}(\mathcal{D})^0$ if $\alpha = \beta$, $\emptyset$ otherwise. By Corollary 6, we know that $\alpha^+(\!|\beta^-|\!)$ reduces either to ε or to 0. The result follows by soundness (Thm. 10). ◀

4.2 (In)equational Full Abstraction

We now show that the operational preorder $\sqsubseteq_{\mathcal{O}}$ (Def. 7) coincides with the inclusion of interpretations in $\mathcal{D}$. The proof of this full abstraction result needs some preliminary lemmas.

▶ **Lemma 17.** *Let $Q \in \Lambda^\tau$, $\mathrm{FV}(Q) \subseteq \vec{x}$ and $\vec{a} \in \mathcal{M}_{\mathrm{f}}(\mathcal{D})^n$. Then $\vec{a} \in [\![Q]\!]_{\vec{x}} \Leftrightarrow [\![Q\langle \vec{a}^-/\vec{x} \rangle]\!] \neq \emptyset$ and $\deg_{x_i}(Q) = \#a_i$.*

Proof. By applying n times (one for each variable in $\vec{x}$) Lemma 8 and Corollary 14. ◀

The ensuing lemma is the key argument for proving that $\mathcal{D}$ is inequationally fully abstract.

▶ **Lemma 18.** *Let $M \in \Lambda^{\bar{\tau}}$, $\vec{x} \supseteq \mathrm{FV}(M)$, $\alpha \in \mathcal{D}$, $\vec{a} \in \mathcal{M}_{\mathrm{f}}(\mathcal{D})$. The following are equivalent:*
(i) $(\vec{a}, \alpha) \in [\![M]\!]_{\vec{x}}$,
(ii) $\alpha^+(\!|M\langle \vec{a}^-/\vec{x} \rangle|\!)\downarrow$.

Proof. We have the following chain of equivalences:

$(\vec{a}, \alpha) \in [\![M]\!]_{\vec{x}} \Leftrightarrow \vec{a} \in [\![\alpha^+(\!|M|\!)]\!]_{\vec{x}}$, by Corollary 14,

$\Leftrightarrow [\![\alpha^+(\!|M\langle \vec{a}^-/\vec{x}\rangle|\!)]\!] \neq \emptyset$ and $\deg_{x_i}(M) = \#a_i$, by Lemma 17, using $(\alpha^+(\!|M|\!))\langle \vec{a}^-/\vec{x}\rangle = \alpha^+(\!|M\langle \vec{a}^-/\vec{x}\rangle|\!)$,

$\Leftrightarrow \alpha^+(\!|M\langle \vec{a}^-/\vec{x}\rangle|\!) \twoheadrightarrow \varepsilon$, by Corollary 6, i.e. the fact that closed tests can only reduce to either ε or 0, and Theorem 10, i.e. the soundness of the model. ◀

▶ **Theorem 19.** *$\mathcal{D}$ is inequationally fully abstract for the $\partial_0\lambda$-calculus with tests:*

$$[\![M]\!]_{\vec{x}} \subseteq [\![N]\!]_{\vec{x}} \Leftrightarrow M \sqsubseteq_{\mathcal{O}} N$$

Proof. ($\Rightarrow$) Assume that $[\![M]\!]_{\vec{x}} \subseteq [\![N]\!]_{\vec{x}}$, and let $C(\!|\cdot|\!)$ be a context closing both M and N and such that $C(\!|M|\!) \twoheadrightarrow \varepsilon$. By Thm. 10, $[\![C(\!|M|\!)]\!] = [\![\varepsilon]\!] = \{()\}$. By monotonicity of the interpretation we get $[\![C(\!|M|\!)]\!] \subseteq [\![C(\!|N|\!)]\!]$, thus $[\![C(\!|N|\!)]\!] \neq \emptyset$. By Cor. 6 this entails $C(\!|N|\!)\downarrow$.

($\Leftarrow$) Suppose, by the way of contradiction, that $M \sqsubseteq_{\mathcal{O}} N$ holds but there is an element $(\vec{a}, \alpha) \in [\![M]\!]_{\vec{x}} - [\![N]\!]_{\vec{x}}$. Then the test-context $C(\!|\cdot|\!) = \alpha^+(\!|(\lambda\vec{x}.(\!|\cdot|\!))\vec{a}^-|\!)$ is such that $C(\!|M|\!) \twoheadrightarrow \alpha^+(\!|M\langle \vec{a}^-/\vec{x}\rangle|\!) \twoheadrightarrow \varepsilon$ and $C(\!|N|\!) \not\twoheadrightarrow \varepsilon$ by Lemma 18. This leads to a contradiction. ◀

The rest of the paper is devoted to extend the above result to the $\partial\lambda$-calculus with tests. The main ingredients will be the Taylor expansion and the head-reduction introduced in Subsections 6.1 and 5.1, respectively.

5 The $\partial\lambda$-Calculus with Tests

The *$\partial\lambda$-calculus with tests* is an extension of the $\partial_0\lambda$-calculus with tests with a promotion available on resources. In this calculus a resource can be linear (it must be used exactly once) or not (it can be used *ad libitum*) and in the latter case it is decorated with a "!" superscript.

Syntax. The grammar generating the terms, the tests and the expressions of the $\partial\lambda$-calculus with tests, is the same as the one for the $\partial_0\lambda$-calculus with tests (in particular tests are still plain multisets of linear resources), except for the rule concerning bags which becomes:

$$P ::= [L_1, \ldots, L_k, \mathbb{N}^!] \qquad \text{bags}$$

where $\mathbb{N}$ is a finite sum of terms of this new syntax. We write $\Lambda_!^{\bar{\tau}}$ for the set of terms generated by this new grammar, $\Lambda_!^{\tau}$ for the set of tests, $\Lambda_!^{b}$ for the set of bags, $\Lambda_!^{e}$ for the set of expressions. From now on bags are no more plain multisets of terms: they are compound objects, consisting of a multiset of terms $[L_1, \ldots, L_k]$ and a sum of terms $\mathbb{N}$, denoted as $[L_1, \ldots, L_k, \mathbb{N}^!]$. We shall deal with them as if they were multisets, defining union by $[L_1, \ldots, L_k, \mathbb{N}^!] \uplus [L_{k+1}, \ldots, L_n, \mathbb{M}^!] := [L_1, \ldots, L_n, (\mathbb{N} + \mathbb{M})^!]$. This operation is commutative, associative and has $[0^!]$ as neutral element.

The $\partial_0\lambda$-calculus with tests is the sub-calculus of $\partial\lambda$-calculus with tests in which all bags have shape $[L_1, \ldots, L_k, 0^!]$, and this identification is compatible with the reduction rules.

As in the $\partial_0\lambda$-calculus with tests, we extend this syntax by multilinearity to sums of expressions with the only exception that the bag $[\vec{L}, (\mathbb{N} + \mathbb{M})^!]$ *is not* required to be equal to $[\vec{L}, \mathbb{N}^!] + [\vec{L}, \mathbb{M}^!]$. The intuition is that in the first expression $\mathbb{N} + \mathbb{M}$ can be used several times and each time one can choose either $\mathbb{N}$ or $\mathbb{M}$, whereas in the second expression one has to choose once and for all one of the summands, and then use it as many times as needed.

Substitutions. Linear substitution is denoted and defined as in the $\partial_0\lambda$-calculus with tests, except of course for bags, where we set:

$$[L_1, .., L_k, \mathbb{N}^!]\langle N/x\rangle = \Sigma_{i=1}^{k}[L_1, .., L_i\langle N/x\rangle, .., L_k, \mathbb{N}^!] + [L_1, .., L_k, \mathbb{N}\langle N/x\rangle, \mathbb{N}^!]\,.$$

For example, $(x[x^!])\langle y/x\rangle\langle z/x\rangle = y[z,x^!] + z[y,x^!] + x[y,z,x^!]$. Remark that in the !-free case, that is when $\mathbb{N} = 0$, the above definitions and notations agree with those introduced in Subsection 2.1, because in that case we have $[L_1,\dots,L_k,\mathbb{N}\langle N/x\rangle,\mathbb{N}^!] = 0$, since $0\langle N/x\rangle = 0$.

We also define the regular substitution $A\{\mathbb{N}/x\}$ for the $\partial\lambda$-calculus with tests, by simply replacing each occurrence of x in the expression A with $\mathbb{N}$: in that way we get an expression of the extended syntax, since $\mathbb{N}$ is a sum in general. E.g., $x[x^!]\{(y+z)/x\} = y[y^!,z^!] + z[y^!,z^!]$.

Both substitutions are then generalized to sums: linear substitution is extended to $\mathbb{A}\langle\mathbb{N}/x\rangle$ by bilinearity in $\mathbb{A}$ and $\mathbb{N}$, while ordinary substitution to $\mathbb{A}\{\mathbb{N}/x\}$ by linearity in $\mathbb{A}$.

A Schwarz lemma, analogous to Lemma 1, holds for the $\partial\lambda$-calculus with tests. Hence, given a sum of expressions $\mathbb{A}$ and a bag $P = [L_1,\dots,L_k]$ with $x \notin \mathrm{FV}(P)$, it still makes sense to set $\mathbb{A}\langle P/x\rangle := \mathbb{A}\langle L_1/x\rangle\cdots\langle L_k/x\rangle$ because this expression does not depend on the enumeration of $L_1,\dots,L_k$. In particular we have $\mathbb{A}\langle[]/x\rangle = \mathbb{A}$.

Operational semantics. The reduction rules of $\partial\lambda$-calculus extend those of the $\partial_0\lambda$-calculus with tests in the sense that they are equivalent on !-free expressions.

The rules (τ) and (γ) are exactly the same, while the β-reduction and $\bar\tau$-reduction are rephrased as follows:

- $(\lambda x.M)[L_1,..,L_k,\mathbb{N}^!] \to_\beta M\langle[L_1,..,L_k]/x\rangle\{\mathbb{N}/x\}$, where wlog $x \notin \mathrm{FV}([L_1,...,L_k])$,
- $\bar\tau(Q)[L_1,\dots,L_k,\mathbb{N}^!] \to_{\bar\tau} \begin{cases} \bar\tau(Q) & \text{if } k=0,\\ 0 & \text{otherwise.}\end{cases}$

The $\partial\lambda$-calculus with tests is still Church-Rosser (just adapt the proof in [14]), while it is no more strongly normalizing. For instance the term $\Omega = (\lambda x.x[x^!])[(\lambda x.x[x^!])^!]$ has an infinite reduction chain, just like the paradigmatic homonymous unsolvable λ-term. Indeed, the usual λ-calculus can be embedded into the $\partial\lambda$-calculus with tests by translating every application MN into $M[N^!]$.

In this framework a *test-context* $C(\!|\cdot|\!)$ is a test of the $\partial\lambda$-calculus with tests having a single occurrence of its *hole*, appearing in term-position. The set of test-contexts is denoted by $\Lambda^{\tau!}_{(\!|\cdot|\!)}$.

A test Q *converges*, notation $Q\downarrow$, if there exists a sum $\mathbb{Q}$ such that $Q \twoheadrightarrow \varepsilon + \mathbb{Q}$.

▶ **Definition 20.** The *operational pre-order* $\sqsubseteq^!_{\mathcal{O}}$ on the $\partial\lambda$-calculus with tests is defined by:

$$M \sqsubseteq^!_{\mathcal{O}} N \Leftrightarrow \forall C(\!|\cdot|\!) \in \Lambda^{\tau!}_{(\!|\cdot|\!)} \text{ closing } M,N\ (C(\!|M|\!)\downarrow \Rightarrow C(\!|N|\!)\downarrow).$$

We then set $M \approx^!_{\mathcal{O}} N$ if and only if $M \sqsubseteq^!_{\mathcal{O}} N$ and $N \sqsubseteq^!_{\mathcal{O}} M$.

Relational semantics. The $\partial\lambda$-calculus with tests can be interpreted into $\mathcal{D}$ by extending the interpretation of the $\partial_0\lambda$-calculus with tests as follows:

$$[\![[L_1,\dots,L_k,\mathbb{N}^!]]\!]_{\vec x} = \{(\uplus_{r=1}^{k+m}\vec a_r,[\beta_1,\dots,\beta_{k+m}]) : (\vec a_j,\beta_j)\in[\![L_j]\!]_{\vec x}, 1\le j\le k \text{ and } (\vec a_i,\beta_i)\in[\![\mathbb{N}]\!]_{\vec x},\ k<i\le k+m\}.$$

It is easy to check that both Lemma 8 and Lemma 9 generalize to this context. From these lemmas it ensues that $\mathcal{D}$ is also a model of the $\partial\lambda$-calculus with tests.

▶ **Theorem 21.** *$\mathcal{D}$ is a model of $\partial\lambda$-calculus with tests.*

5.1 Head Reduction

We now provide a notion of *head-reduction* for the $\partial\lambda$-calculus with tests. Intuitively, the head-reduction is obtained by reducing a head-redex, that is a redex occurring in head-position in an expression A. The interest of introducing this reduction strategy is that it "behaves well" with respect to the Taylor expansion in the sense of Proposition 31.

We start by defining the notion of redex.

$$x^\circ = \{x\}, \qquad (\lambda x.M)^\circ = \{\lambda x.M' : M' \in M^\circ\}, \qquad (MP)^\circ = \{M'P' : M' \in M^\circ,\ P' \in P^\circ\},$$
$$(\bar\tau(Q))^\circ = \{\bar\tau(Q') : Q' \in Q^\circ\}, \quad (\tau[M_1,\dots,M_k])^\circ = \{\tau[M'_1,\dots,M'_k] : M'_i \in M_i^\circ,\ \text{for } 1 \le i \le k\},$$
$$[L_1,\dots,L_k,\mathbb{N}^!]^\circ = \{[L'_1,\dots,L'_k] \uplus P : L'_i \in L_i^\circ,\ \text{for } 1 \le i \le k,\ P \in \mathcal{M}_{\mathrm{f}}(\mathbb{N}^\circ)\},$$
$$(\Sigma_{i=1}^k A_i)^\circ = \cup_{i=1}^k A_i^\circ.$$

Figure 1 The *Taylor expansion* $\mathbb{A}^\circ$ of $\mathbb{A} \in \mathbf{2}\langle\Lambda^e_!\rangle$.

▶ **Definition 22.** A *term-redex* is any term of the form $(\lambda x.M)P$ or $\bar\tau(Q)P$. A *test-redex* is any test of the form $\tau[\lambda x.M]$, $\tau[\bar\tau(Q)]$.

Among term- and test-redexes we distinguish those redexes that are in "head" position.

▶ **Definition 23.** A *head-redex* is:
- either a term-redex H in terms of shape $\lambda\vec{y}.H\vec{P}$,
- or a term-redex H in tests of shape $\tau[H\vec{P}]|Q$,
- or a test-redex R in tests of shape $R|Q$.

▶ **Definition 24.** We say that $A \to \mathbb{B}$ is a step of *head-reduction* if $\mathbb{B}$ is obtained from A by contracting a head-redex. If $A \to \mathbb{B}$ is a step of head-reduction then also $A + \mathbb{A} \to \mathbb{B} + \mathbb{A}$ is.

One-step head-reduction is denoted by $\to_h$, while $\twoheadrightarrow_h$ indicates its reflexive and transitive closure. Notice that, unlike in ordinary λ-calculus, an expression A may have more than one head-redex, hence there may be more than one head-reduction steps starting from A.

The head-reduction induces a notion of head-normal form on (finite sums of) expressions.

▶ **Definition 25.** An expression A *is in head-normal form* (*hnf*, for short) if there is no $\mathbb{B}$ such that $A \to_h \mathbb{B}$; a sum $\mathbb{A}$ *is in hnf* if $\mathbb{A} \neq 0$ and each summand is in hnf.

This notion of hnf differs from that given by Pagani and Ronchi della Rocca in [13]. We keep this name since their definition captures the notion of "outer-nf" rather than that of hnf.

It is easy to check that a term M is in hnf iff $M := \lambda\vec{x}.y\vec{P}$ or $M := \lambda\vec{x}.\bar\tau(Q)$; a test R is in hnf iff $R := \varepsilon$, $R := \tau[x\vec{P}]$ or $R := Q_1|Q_2$ for some tests Q_1, Q_2 in hnf.

The following two lemmas concern reduction properties of !-*free closed* tests.

▶ **Lemma 26.** *Let $R \in \Lambda^\tau$. If R is closed and $R \neq \varepsilon$ then it has a head-redex (hence, $R \to_h \mathbb{R}'$ for some $\mathbb{R}'$).*

Proof. By induction on R. It suffices to consider the case $R = \tau[M]$. We then proceed by cases on the structure of M (which must be closed). If $M = \lambda x.N$ then R head-reduces using (τ). If M is an application then it must be written either as $M = (\lambda y.N)P_1 \cdots P_k$ or as $M = \bar\tau[Q]P_1 \cdots P_k$ (in both cases $k \ge 1$) and hence R head-reduces using either (β) or $(\bar\tau)$, respectively. If $M = \bar\tau(Q)$ then R head-reduces using (γ). ◀

▶ **Lemma 27.** *If $R \in \Lambda^\tau$ is closed then $R \twoheadrightarrow \varepsilon$ iff $R \twoheadrightarrow_h \varepsilon$.*

Proof. ($\Rightarrow$) Suppose, by contradiction, that $R \twoheadrightarrow \varepsilon$ but $R \not\twoheadrightarrow_h \varepsilon$. By confluence (Thm. 4), we cannot have $R \twoheadrightarrow_h 0$. Thus, since $R \in \Lambda^\tau$ is strongly normalizing, the only way to have $R \not\twoheadrightarrow_h \varepsilon$ is that $R \twoheadrightarrow_h \mathbb{R}$ where $\mathbb{R} \neq \varepsilon$ is in hnf. This is impossible by Lemma 26.

($\Leftarrow$) Trivial since $\twoheadrightarrow_h \subseteq \twoheadrightarrow$. ◀

One should be careful when trying to extend the above result to terms $M \in \Lambda^{\bar\tau}$. For instance, it is false that $M \twoheadrightarrow 0$ iff $M \twoheadrightarrow_h 0$. Indeed $M := \lambda x.x[\mathbf{I}[]]$ is in hnf but $M \to \lambda x.x[0] := 0$.

6 Full Abstraction via Taylor Expansion

In this section we are going to define the Taylor expansion of terms and tests of the $\partial\lambda$-calculus with tests. We will then use this expansion, combined with head-reduction, to generalize the full abstraction results obtained in Subsection 4.2 to the framework of $\partial\lambda$-calculus with tests.

6.1 Taylor Expansion

The *(full) Taylor expansion* was first introduced in [6, 7], in the context of λ-calculus. The Taylor expansion M° of an ordinary λ-term M gives an infinite formal linear combination of terms of the $\partial_0\lambda$-calculus. In the case of ordinary application it looks like:

$$(MN)^\circ = \sum_{n=0}^{\infty} \frac{1}{n!} M[\underbrace{N,\ldots,N}_{n \text{ times}}]$$

in accordance with the intended meaning and the denotational semantics of application in the resource calculus. In the syntax of differential λ-calculus [6] the above formula looks like $\sum_{n=0}^{\infty} \frac{1}{n!} M^{(n)}(0)(N,\ldots,N)$, hence the connection with analytical Taylor expansion is clear.

Following [10], we extend the definition of Taylor expansion from ordinary λ-terms to the expressions of the $\partial\lambda$-calculus with tests. Since the sum is idempotent, the coefficients disappear and our Taylor expansion corresponds to the *support* of the actual Taylor expansion.

As the set $\mathbf{2}\langle\Lambda^e\rangle_\infty$ of possibly infinite formal sums of expressions is isomorphic to $\mathcal{P}(\Lambda^e)$, in the following we may use sets instead of sums.

▶ **Definition 28.** Let $\mathbb{A} \in \mathbf{2}\langle\Lambda^e_!\rangle$. The *(full) Taylor expansion of* $\mathbb{A}$ is the set $\mathbb{A}^\circ \subseteq \Lambda^e$ which is defined (by structural induction on $\mathbb{A}$) in Figure 1.

As previously announced, the Taylor expansion of an expression A can be infinite. For example, we have that $(\lambda x.x[x^!])^\circ = \{\lambda x.x[x^n] : n \in \mathbf{N}\}$.

To lighten the notations, we adopt for infinite sets of expressions the same abbreviations as introduced for finite sums in Subsection 2.1 (including those for substitutions). For instance, if $X, Y \subseteq \Lambda^{\bar\tau}$ then $\lambda x.X$ denotes the set $\{\lambda x.M' : M' \in X\}$ and $X\langle Y/x\rangle = \cup_{M\in X, N\in Y} M\langle N/x\rangle$.

In [10] it is proved that **MRel** is a differential Cartesian closed category that "models the Taylor expansion". This property entails that Taylor expansion preserves the meaning of an expression in $\mathcal{D}$, as expressed in the next theorem.

▶ **Theorem 29.** $[\![\mathbb{A}]\!]_{\vec{x}} = \cup_{A\in\mathbb{A}^\circ} [\![A]\!]_{\vec{x}}$, *for all* $\mathbb{A} \in \mathbf{2}\langle\Lambda^e_!\rangle$.

Proof. By adapting the proof in [10] of the analogous result for the differential λ-calculus. ◀

We now need the following lemma stating the commutation of Taylor expansion with respect to ordinary and linear substitutions. The proof is lengthy but not difficult and is omitted. For the sake of readability, in the next statements we use sums and unions interchangeably.

▶ **Lemma 30.** *Let* $A \in \Lambda^e_!$, $N \in \Lambda^{\bar\tau}_!$ *and* $\mathbb{N} \in \mathbf{2}\langle\Lambda^{\bar\tau}_!\rangle$. *Then, for* $x \notin \mathrm{FV}(N) \cup \mathrm{FV}(\mathbb{N})$*:*

(i) $(A\langle N/x\rangle)^\circ = A^\circ\langle N^\circ/x\rangle$,

(ii) $(A\{\mathbb{N}/x\})^\circ = \bigcup_{P\in\mathcal{M}_{\mathrm{f}}(\mathbb{N}^\circ)} A^\circ\langle P/x\rangle\{0/x\}$.

The next proposition is devoted to show how Taylor expansion interacts with head-reduction. To ease the formulation of the next proposition we assimilate $\mathbf{2}\langle\Lambda^e_!\rangle$ to $\mathcal{P}_{\mathrm{f}}(\Lambda^e_!)$.

▶ **Proposition 31.** *Let* $A \in \Lambda^e_!$ *and let* $A' \in A^\circ$ *be such that* $A' \to_h \mathbb{B}'$, *for some* $\mathbb{B}'$. *Then there exists* $\mathbb{B}$ *such that* $A \to_h \mathbb{B}$ *and* $\mathbb{B}' \subseteq \mathbb{B}^\circ$.

Proof. The idea is that the syntactic tree of A has the same structure as that of A' and we can define a surjective mapping of the redexes of A' into those of A.

We only treat the case $A' = \lambda\vec{x}.H'P'_1\cdots P'_p$ where $H' = (\lambda y.M')P'$ is a head-redex. From $A' \in A^\circ$ we get $A = \lambda\vec{x}.HP_1\cdots P_p$ for some H such that $H' \in H^\circ$. Hence, supposing wlog $P' = [\vec{L}', \vec{N}']$, we have that $H = (\lambda y.M)[\vec{L}, \mathbb{N}^!]$ where $M' \in M^\circ$, the lengths of $\vec{L}'$ and $\vec{L}$ coincide, $L'_i \in L_i^\circ$ for all i and $[\vec{N}'] \in \mathcal{M}_{\mathrm{f}}(\mathbb{N}^\circ)$. We now know that $H' \to_h M'\langle[\vec{L}']/y\rangle\langle[\vec{N}']/y\rangle\{0/y\}$ and $H \to_h M\langle[\vec{L}]/y\rangle\{\mathbb{N}/y\}$. By Lemma 30, $(M\langle[\vec{L}]/y\rangle\{\mathbb{N}/y\})^\circ = \cup_{P\in\mathcal{M}_{\mathrm{f}}(\mathbb{N}^\circ)} M^\circ\langle[\vec{L}^\circ]/y\rangle\langle P/y\rangle\{0/y\} \supseteq M\langle P'/y\rangle\{0/y\}$.

We can conclude that $\lambda\vec{x}.M'\langle P'/y\rangle\{0/y\}P'_1\cdots P'_p \subseteq (\lambda\vec{x}.M\langle[\vec{L}]/y\rangle\{\mathbb{N}/y\}P_1\cdots P_p)^\circ$. ◀

Note that the above proposition is false for regular β-reduction. E.g., take $A := x[(\mathbf{I}[y])^!]$ and $A' := x[\mathbf{I}[y], \mathbf{I}[y]] \in A^\circ$, then $A' \to_\beta x[y, \mathbf{I}[y]]$ and $A \to_\beta x[y^!]$ but $x[y, \mathbf{I}[y]] \notin (x[y^!])^\circ$.

▶ **Corollary 32.** *Let $R \in \Lambda_!^\tau$ be closed. If there is an $R' \in R^\circ$ such that $R' \twoheadrightarrow \varepsilon$, then $R{\downarrow}$.*

Proof. Suppose that there exists $R' \in R^\circ$ such that $R' \twoheadrightarrow \varepsilon$. By Lemma 27 there is a head-reduction chain of the form $R' \to_h \mathbb{R}'_1 \to_h \cdots \to_h \mathbb{R}'_n = \varepsilon$. By iterated application of a corollary[2] of Prop. 31 there are tests $\mathbb{R}_i$ (for $i = 1, \ldots, n$) such that $R \to_h \mathbb{R}_1 \to_h \cdots \to_h \mathbb{R}_n$ with $\mathbb{R}'_i \subseteq \mathbb{R}_i^\circ$. We conclude since $\varepsilon \in \mathbb{R}_n^\circ$ is only possible when $\varepsilon \in \mathbb{R}_n$. ◀

6.2 Full Abstraction for the $\partial\lambda$-Calculus with Tests

We now prove that the relational model $\mathcal{D}$ is fully abstract for the $\partial\lambda$-calculus with tests.

▶ **Lemma 33.** *Given $A \in \Lambda_!^e$ and $M \in \Lambda_!^{\bar{\tau}}$ we have:*

(i) $(\alpha^+(\!|M|\!))^\circ = \alpha^+(\!|M^\circ|\!)$, *for all* $\alpha \in \mathcal{D}$,
(ii) $(A\langle a^-/x\rangle)^\circ = A^\circ\langle a^-/x\rangle$, *for all* $a \in \mathcal{M}_{\mathrm{f}}(\mathcal{D})$.

Proof. As $\alpha^+(\!|\cdot|\!)$ and a^- are !-free, and $(\cdot)^\circ$ behaves like the identity on !-free expressions. ◀

▶ **Proposition 34.** *Let $M \in \Lambda_!^{\bar{\tau}}$, $\vec{x} \supseteq \mathrm{FV}(M)$, $\alpha \in \mathcal{D}$ and $\vec{a} \in \mathcal{M}_{\mathrm{f}}(\mathcal{D})$. Then the following statements are equivalent:*

(i) $(\vec{a}, \alpha) \in [\![M]\!]_{\vec{x}}$,
(ii) $\alpha^+(\!|M\langle\vec{a}^-/\vec{x}\rangle|\!){\downarrow}$.

Proof. (i ⇒ ii) Suppose $(\vec{a}, \alpha) \in [\![M]\!]_{\vec{x}}$, then by Thm. 29 there is an $M' \in M^\circ$ such that $(\vec{a}, \alpha) \in [\![M']\!]_{\vec{x}}$. Applying Lemma 18 we know that $\alpha^+(\!|M'\langle\vec{a}^-/\vec{x}\rangle|\!) \twoheadrightarrow \varepsilon$. Now, since $\alpha^+(\!|M'\langle\vec{a}^-/\vec{x}\rangle|\!) \in (\alpha^+(\!|M\langle\vec{a}^-/\vec{x}\rangle|\!))^\circ$ (by Lemma 33), we can apply Corollary 32 and get $\alpha^+(\!|M\langle\vec{a}^-/\vec{x}\rangle|\!){\downarrow}$.

(ii ⇒ i) Suppose that $\alpha^+(\!|M\langle\vec{a}^-/\vec{x}\rangle|\!) \twoheadrightarrow \varepsilon + \mathbb{Q}$, for some $\mathbb{Q}$; then $[\![\alpha^+(\!|M\langle\vec{a}^-/\vec{x}\rangle|\!)]\!]_{\vec{x}} \neq \emptyset$. Hence, by Theorem 29, there is a closed test $R \in (\alpha^+(\!|M\langle\vec{a}^-/\vec{x}\rangle|\!))^\circ$ such that $[\![R]\!] \neq \emptyset$. By Lemma 33 $R = \alpha^+(\!|M'\langle\vec{a}^-/\vec{x}\rangle|\!)$ for some $M' \in M^\circ$ and since its interpretation is non-empty we have $R \twoheadrightarrow \varepsilon$. By applying Lemma 18 we get $(\vec{a}, \alpha) \in [\![M']\!]_{\vec{x}} \subseteq [\![M]\!]_{\vec{x}}$ (by Theorem 29). ◀

▶ **Theorem 35.** *$\mathcal{D}$ is inequationally fully abstract for the $\partial\lambda$-calculus with tests:*

$$[\![M]\!]_{\vec{x}} \subseteq [\![N]\!]_{\vec{x}} \Leftrightarrow M \sqsubseteq^!_{\mathcal{O}} N.$$

[2] If $\mathbb{A}' \subseteq \mathbb{A}^\circ$ and $\mathbb{A}' \to_h \mathbb{B}'$ then there exists $\mathbb{B}$ such that $\mathbb{A} \to_h \mathbb{B}$ and $\mathbb{B}' \subseteq \mathbb{B}^\circ$.

Proof. ($\Rightarrow$) Suppose that $[\![M]\!]_{\vec{x}} \subseteq [\![N]\!]_{\vec{x}}$ and there is a test-context $C(\!|\cdot|\!)$ (closing M, N) such that $C(\!|M|\!)\downarrow$. Since $C(\!|M|\!) \twoheadrightarrow \varepsilon + \mathbb{Q}$, for some $\mathbb{Q}$, we have $[\![C(\!|M|\!)]\!] \neq \emptyset$. Thus, by monotonicity of the interpretation we get $[\![C(\!|M|\!)]\!] \subseteq [\![C(\!|N|\!)]\!] = [\![(C(\!|N|\!))^\circ]\!] \neq \emptyset$. By Corollary 6 there is $R \in (C(\!|N|\!))^\circ$ such that $R \twoheadrightarrow \varepsilon$ and we conclude that $C(\!|N|\!)\downarrow$ by applying Proposition 34.

($\Leftarrow$) Suppose by contradiction that $M \sqsubseteq^!_{\mathcal{O}} N$, but there is an $(\vec{a}, \alpha) \in [\![M]\!]_{\vec{x}} - [\![N]\!]_{\vec{x}}$. By Prop. 34 $\alpha^+(\!|M\langle \vec{a}^-/\vec{x}\rangle|\!)\downarrow$ and since $M \sqsubseteq^!_{\mathcal{O}} N$ we have $\alpha^+(\!|N\langle \vec{a}^-/\vec{x}\rangle|\!)\downarrow$. Again, by Prop. 34 $(\vec{a}, \alpha) \in [\![N]\!]_{\vec{x}}$. Contradiction. ◀

Further Work. We proved that $\mathcal{D}$ is a fully abstract model of the $\partial\lambda$-calculus and of the $\partial_0\lambda$-calculus with tests. We strongly conjecture that it also (in)equationally fully abstract for the corresponding calculi *without tests*. A possible approach to obtain these results might be to define a "test-expansion" translating every test-context $C(\!|\cdot|\!)$ sending $M \in \Lambda^{\vec{!}}_!$ to $\varepsilon + \mathbb{R}$ into a term-context $C'(\!|\cdot|\!)$ sending M to $\mathbf{I} + \mathbb{N}$. This generalization is non trivial and is kept for future work. Another open problem is to find a fully abstract model of these calculi where $+$ is treated as *must* non-determinism (a sum converges if all its summands converge).

References

1 H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, Amsterdam, 1984.

2 G. Boudol. The lambda-calculus with multiplicities (abstract). In Eike Best, editor, *CONCUR*, volume 715 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 1993.

3 G. Boudol, P.-L. Curien, and C. Lavatelli. A semantics for lambda calculi with resources. *Math. Struct. in Comp. Sci.*, 9(4):437–482, 1999.

4 A. Bucciarelli, T. Ehrhard, and G. Manzonetto. Not enough points is enough. In *CSL'07*, volume 4646 of *LNCS*, pages 298–312. Springer, 2007.

5 T. Ehrhard and O. Laurent. Interpreting a finitary pi-calculus in differential interaction nets. *Inf. Comput.*, 208(6):606–633, 2010.

6 T. Ehrhard and L. Regnier. The differential lambda-calculus. *Theor. Comput. Sci.*, 309(1-3):1–41, 2003.

7 T. Ehrhard and L. Regnier. Böhm trees, Krivine's machine and the Taylor expansion of lambda-terms. In *CiE*, volume 3988 of *LNCS*, pages 186–197. Springer, 2006.

8 T. Ehrhard and L. Regnier. Uniformity and the Taylor expansion of ordinary lambda-terms. *Theor. Comput. Sci.*, 403(2-3):347–372, 2008.

9 J.-Y. Girard. From foundations to ludics. *Bulletin of Symbolic Logic*, 9(2):131–168, 2003.

10 G. Manzonetto. What is a categorical model of the differential and the resource lambda calculi? June 2010. Submitted to Mathematical Structures in Computer Science.

11 R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, 1982.

12 R. Milner. The polyadic π-calculus: a tutorial. *Logic and algebra of specification*, pages 203–246, 1993.

13 M. Pagani and S. Ronchi Della Rocca. Solvability in resource lambda-calculus. In *Proc. of FOSSACS'10*, volume 6014 of *LNCS*, pages 358–373. Springer, 2010.

14 M. Pagani and P. Tranquilli. Parallel reduction in resource lambda-calculus. In *APLAS'09*, volume 5904 of *LNCS*, pages 226–242. Springer, 2009.

15 P. Tranquilli. Intuitionistic differential nets and lambda-calculus. *Theor. Comp. Sci.* To appear.

16 L. Vaux. The differential $\lambda\mu$-calculus. *Theor. Comput. Sci.*, 379(1-2):166–209, 2007.

Tight Upper Bounds for Streett and Parity Complementation*

Yang Cai[1] and Ting Zhang[2]

1 **MIT Computer Science and Artificial Intelligence Laboratory**
The Stata Center, 32-G696, Cambridge, MA 02139 USA
ycai@csail.mit.edu

2 **Department of Computer Science, Iowa State University**
226 Atanasoff Hall, Ames, IA 50011 USA
tingz@iastate.edu

Abstract

Complementation of finite automata on infinite words is not only a fundamental problem in automata theory, but also serves as a cornerstone for solving numerous decision problems in mathematical logic, model-checking, program analysis and verification. For Streett complementation, a significant gap exists between the current lower bound $2^{\Omega(n \lg nk)}$ and upper bound $2^{O(nk \lg nk)}$, where n is the state size, k is the number of Streett pairs, and k can be as large as 2^n. Determining the complexity of Streett complementation has been an open question since the late 80's. In this paper we show a complementation construction with upper bound $2^{O(n \lg n + nk \lg k)}$ for $k = O(n)$ and $2^{O(n^2 \lg n)}$ for $k = \omega(n)$, which matches well the lower bound obtained in [3]. We also obtain a tight upper bound $2^{O(n \lg n)}$ for parity complementation.

1998 ACM Subject Classification F.1.1 Models of Computation; F.4.1 Mathematical Logic; F.4.3 Formal Languages

Keywords and phrases Streett automata, ω-automata, parity automata, complementation, upper bounds

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.112

1 Introduction

Automata on infinite words (ω-automata) have wide applications in synthesis and verification of reactive concurrent systems. Complementation plays a fundamental role in many of these applications, especially in solving the language containment problem: whether a language recognized by automaton $\mathcal{A}$ is contained by another language represented by automaton $\mathcal{B}$, which is equivalent to whether the language of $\mathcal{A}$ and the complementary language of $\mathcal{B}$ intersect. In automata-theoretic model checking [10, 27], both system behaviors and logical specifications are represented as formal languages, and model checking by and large amounts to solving the corresponding language containment problem. As both language intersection and emptiness test are rather easy, the efficiency of complementation becomes crucial to practical deployment of model-checking tools. For this reason and many others, determining the state complexity of the complementation problem has been extensively studied in the last four decades [26].

* This research has been supported by NSF CAREER Award CCF-0954132.

Related Work

ω-automata were invented by Büchi in 1962 as a method of attack on definability and decision problems for monadic second order logic on arithmetics (S1S) [1]. That type of ω-automata are nowadays called Büchi automata. The initial Büchi complementation was not explicitly constructive and required double exponential blow-up [1]. But since then Büchi complementation has been extensively studied. The upper bound was continuously improved to $2^{O(n^2)}$ [24], $2^{O(n \lg n)}$ [21], $O((6n)^n)$ [11], $O((0.97n)^n)$ [6] and finally to $O(n^2 L(n))$ where $L(n) \approx (0.76n)^n$ [23], which matched well the lower bound $\Omega(L(n))$ [28].

Complementation for automata with rich acceptance conditions, such as Rabin automata and Streett automata, is much more sophisticated. Kupferman and Vardi showed a $2^{O(nk \lg n)}$ complementation construction for Rabin automata [14], and we showed this construction is essentially optimal [2]. This leaves Streett complementation the last classical problem where the gap between the lower and upper bounds is substantial. Besides that, Streett complementation has an importance of its own. Streett automata share identical algebraic structures with Büchi automata, except being equipped with richer acceptance conditions. A Streett acceptance condition comprises a finite list of indexed pairs of sets of states. Each pair consists of an enabling set and a fulfilling set. A run is accepting if for each pair, if the run visits states in the enabling set infinitely often, then it also visits states in the fulfilling set infinitely often. This naturally corresponds to the *strong fairness* condition that infinitely many requests are responded infinitely often, a necessary requirement for meaningful computations [5, 7]. Another advantage of Streett automata is that they are much more succinct than Büchi automata; it is unavoidable in the worst case to have 2^n state blow-up to translate a Streett automaton with $O(n)$ states and $O(n)$ index pairs to an equivalent Büchi automaton [25]. An interesting question is: to what extent does the gain from the succinctness have to be paid back at the time of complementation?

The first construction for Streett complementation was given by Safra and Vardi, and that construction required $2^{O((nk)^5)}$ state blow-up [25]. Klarlund improved this bound to $2^{O(nk \lg nk)}$ [8]. The same bound was achieved by Safra via determinization [22], by Piterman with an improved determinization construction [19], and by Kupferman and Vardi [14]. However, so far no construction has been proved to cost less than $2^{O(nk \lg nk)}$ states. The question of whether Streett complementation can be further improved from $2^{O(nk \lg nk)}$ has been constantly raised in the recent literature [14, 28, 26]. In this paper we answer this question affirmatively.

Ranking-based Complementation

A Ranking-based complementation was first proposed by Klarlund [8]. Klarlund's Büchi complementation (resp. Streett complementation) relies on *quasi co-Büchi measure* (resp. *quasi Rabin measure*), which is a ranking function on states in a run graph, measuring the progress of a run toward being accepted. By this complementation scheme, Klarlund gave a $2^{O(n \lg n)}$ Büchi complementation and a $2^{O(nk \lg nk)}$ Streett complementation [8]. Kupferman and Vardi developed a similar idea into an elegant and comprehensive framework [13, 9], obtaining complementation constructions for Büchi [11], generalized Büchi [12], Rabin and Streett [14].

Our Results

Our Streett complementation is obtained by improving Kupferman and Vardi's construction in [14]. We show that the larger the Rabin index size k, the higher the correlation between

Type	Bound		Lower	Upper
Büchi	$2^{\Theta(n \lg n)}$		[17]	[21]
Generalized Büchi	$2^{\Theta(n \lg nk)}$	$k = O(2^n)$	[28]	[14]
Streett	$2^{\Theta(n \lg n + nk \lg k)}$	$k = O(n)$	[3]	this
	$2^{\Theta(n^2 \lg n)}$	$k = \omega(n)$		
Rabin	$2^{\Theta(nk \lg n)}$	$k = O(2^n)$	[2]	[14]
Parity	$2^{\Theta(n \lg n)}$	$k = O(n)$	[17]	this

Figure 1 The complementation complexities for ω-automata of common types.

infinite paths in a run graph satisfying a universal Rabin condition (the dual of an existential Streett condition), and characterize the correlation using two tree structures: ITS (*Increasing Tree of Sets*) and TOP (*Tree of Ordered Partitions*), both with elegant combinatorial properties. We show that our construction renders a upper bound $U(n, k)$, which is $2^{O(n \lg n + nk \lg k)}$ for $k = O(n)$ and $2^{O(n^2 \lg n)}$ for $k = \omega(n)$. $U(n, k)$ is a significant improvement from the previous best bound when $k = \omega(n)$. Speaking loosely, we gain succinctness without paying a dramatically higher price for complementation. $U(n, k)$ also matches the lower bound $L(n, k)$, which is $2^{\Omega(n \lg n + nk \lg k)}$ for $k = O(n)$ and $2^{\Omega(n^2 \lg n)}$ for $k = \omega(n)$ [3]. By a similar technique, we also obtain a $2^{O(n \lg n)}$ upper bound for parity complementation, which is essentially optimal, as parity automata generalizes Büchi automata, whose complementation lower bound is $2^{\Omega(n \lg n)}$ [17, 15]. This is surprising as the index size k (though small as $k \leq \lfloor (n + 1)/2 \rfloor$) has no appearance in the asymptotical bound. We believe this is of practical interest as well, because it tells us that parity automata provide a richer acceptance condition without incurring an asymptotically higher cost on complementation. Combining the result with the one in [3] and previous findings in the literature, we now have a complete characterization of complementation complexity for ω-automata of common types. Figure 1 summarizes these results.

Paper Organization

Section 2 introduces basic notations and terminology in automata theory. Section 3 presents the framework of ranking based complementation; it introduces Büchi complementation [11], generalized Büchi complementation [12] and Streett complementation [14]. Section 4 presents our Streett complementation construction and Section 5 proves its complexity. Section 6 establishes a tight upper bound for parity complementation. Section 7 concludes with a discussion of future work. Due to space limit, all proofs are omitted, but they can be found in the full version of this paper at arXiv:1102.2960.

2 Preliminaries

Basic Notations

Let $\mathbb{N}$ denote the set of natural numbers. We write $[i..j]$ for $\{k \in \mathbb{N} \mid i \leq k \leq j\}$, $[i..j)$ for $[i..j-1]$, $[n]$ for $[0..n)$, and $[n]^{\text{even}}$ and $[n]^{\text{odd}}$ for even numbers and odd numbers in $[n]$, respectively. For an infinite sequence ϱ, we use $\varrho(i)$ to denote the i-th component for $i \in \mathbb{N}$. For a finite sequence α, we use $|\alpha|$ to denote the length of α, $\alpha[i]$ $(i \in [1..|\alpha|])$ to denote the object at the i-th position, and $\alpha[i..j]$ (resp. $\alpha[i..j)$) to denote the subsequence of α from position i to position j (resp. $j - 1$). When we compare finite sequences of

numbers, $>_m, \geq_m, =_m, <_m, \leq_m$ mean the corresponding standard lexicographical orderings up to position m. We reserve n and k as parameters of complementation instances (n for state size and k for index size), and define $\mu = \min(n, k)$ and $I = [1..k]$.

Automata and Runs.

A finite automaton on infinite words (ω-automaton) is a tuple $\mathcal{A} = (\Sigma, Q, Q_0, \Delta, \mathcal{F})$ where Σ is an alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\Delta \subseteq Q \times \Sigma \times Q$ is a set of transitions, and $\mathcal{F}$ is an acceptance condition.

An infinite word (ω-word) over Σ is an infinite sequence of letters in Σ. A *run* ϱ of $\mathcal{A}$ over an ω-word w is an infinite sequence of states in Q such that $\varrho(0) \in Q_0$ and, $\langle \varrho(i), w(i), \varrho(i{+}1) \rangle \in \Delta$ for $i \in \mathbb{N}$. Let $\mathrm{Inf}(\varrho)$ be the set of states that occur infinitely many times in ϱ. An automaton accepts w if a run ϱ over w exists that satisfies $\mathcal{F}$, which is usually defined as a predicate on $\mathrm{Inf}(\varrho)$. The language of $\mathcal{A}$, written $\mathscr{L}(\mathcal{A})$, is the set of ω-words accepted by $\mathcal{A}$.

Acceptance Conditions and Types

ω-automata are classified according to their acceptance conditions. Below we list automata of common types. Let G and B be functions from I to 2^Q.

- *Generalized Büchi*: $\langle B \rangle_I$: $\forall i \in I, \mathrm{Inf}(\varrho) \cap B(i) \neq \emptyset$.
- *Büchi*: $\langle B \rangle_I$ with $I = \{1\}$ (i.e., $k = 1$).
- *Streett*: $\langle G, B \rangle_I$: $\forall i \in I$, $\mathrm{Inf}(\varrho) \cap G(i) \neq \emptyset \rightarrow \mathrm{Inf}(\varrho) \cap B(i) \neq \emptyset$.
- *Parity*: $\langle G, B \rangle_I$ with $B(1) \subset G(1) \subset \cdots \subset B(k) \subset G(k)$.
- *Generalized co-Büchi*: $[B]_I$: $\exists i \in I$, $\mathrm{Inf}(\varrho) \cap B(i) = \emptyset$.
- *Co-Büchi*: $[B]_I$ with $I = \{1\}$ (i.e., $k = 1$).
- *Rabin*: $[G, B]_I$: $\exists i \in I$, $\mathrm{Inf}(\varrho) \cap G(i) \neq \emptyset \wedge \mathrm{Inf}(\varrho) \cap B(i) = \emptyset$.

We use GB, B, S, P, GC, CB, and R, respectively, to denote the above acceptance conditions. By T-automata we mean the ω-automata with T-condition. Note that B and CB, GB and GC, and S and R are dual to each other, respectively. Also note that generalized Büchi and parity automata are both subclasses of Streett automata, and so are generalized co-Büchi and parity automata to Rabin automata. Let $J \subseteq I$. We use $[G, B]_J$ to denote the Rabin condition with respect to only indices in J. When J is a singleton, say $J = \{j\}$, we simply write $[G(j), B(j)]$ for $[G, B]_J$. The same convention is used for other conditions. For a Streett condition $\langle G, B \rangle_I$, we can assume that B is injective, because if $B(i) = B(i')$ for two different $i, i' \in I$, then we can replace $\langle G, B \rangle_{\{i,i'\}}$ by $\langle G(i) \cup G(i'), B(i) \rangle$. The same assumption is made for any Rabin condition $[G, B]_I$.

Δ-Graphs.

A Δ-*graph* of an ω-word w under $\mathcal{A}$ is a directed graph $\mathscr{G}_w = (V, E)$ where $V = Q \times \mathbb{N}$ and $E = \{\langle \langle q, l \rangle, \langle q', l+1 \rangle \rangle \in V \times V \mid q, q' \in Q,\ i \in \mathbb{N}, \langle q, w(i), q' \rangle \in \Delta\}$. By the *i-th level*, we mean the vertex set $Q \times \{i\}$. Let S be a subset of Q. We call a vertex $v = \langle q, l \rangle$ *S-vertex* if $q \in S$. When level index is of no importance in the context, we use q and v interchangeably. In particular, by an abuse of notation we write $v \in S$ to mean $v = \langle q, l \rangle$ for some $l \in \mathbb{N}$ and $q \in S$. Δ-graphs for finite words are similarly defined. The length of a finite Δ-graph is the number of levels minus 1. By unit Δ-graphs we mean Δ-graphs of length 1. A unit Δ-graph encodes all possible transitions upon reading a letter. By *width* of $\mathscr{G}_w$ (written *width*($\mathscr{G}_w$))

we mean the maximum number of pairwise non-intersecting infinite paths in $\mathscr{G}_w$. Clearly, for any w, $width(\mathscr{G}_w) \leq |Q|$.

3 Ranking-based Complementation

In this section we introduce ranking-based complementation constructions developed by Klarlund, Kupferman and Vardi [8, 11, 13, 9]. Note that all complexity related notions are parameterized with n and k, but we do not list them explicitly unless required for clarity. We adopt the following naming convention: when we talk about behaviors of a source automaton, a T-condition means an *existential* one (i.e., a path in a Δ-graph that satisfies T), while in the context of complementation, a T-condition means a *universal* one (i.e., every path in a Δ-graph satisfies T).

Ranking-based Complementation Scheme

Let $\mathcal{A}$ be a T-automaton and $\mathcal{CA}$ a purported Büchi automaton that complements $\mathcal{A}$. An ω-word w is accepted by $\mathcal{A}$ if and only if the Δ-graph $\mathscr{G}_w$ contains an infinite path that satisfies the T-condition. Consequently, w is accepted by $\mathcal{CA}$ if and only if *all* paths in $\mathscr{G}_w$ satisfy the dual co-T condition (for short, $\mathscr{G}_w$ is co-T accepting). Complementation essentially amounts to transforming a *universal* co-T condition into an *existential* Büchi condition. Rankings on Δ-graphs provide a solution; $\mathscr{G}_w$ satisfying a universal co-T condition is precisely captured by the existence of a so-called odd co-T ranking on $\mathscr{G}_w$. Complementation then reduces to recognition of Δ-graphs that admit odd co-T rankings.

The general scheme goes as follows. Vertices of $\mathscr{G}_w$ are associated with certain values. The association at a level can be viewed as a function with domain Q (with level indices dropped), called *co-T level ranking*. The values in the range of a co-T level ranking are called *co-T ranks*, and the n-tuple of co-T ranks at a level is called a *co-T level rank*. By a *co-T ranking* we mean an ω-sequence of co-T level rankings, each of which is associated with a level in $\mathscr{G}_w$. Co-T rankings are required to satisfy a *local* property, which holds between every two adjacent levels and is *solely* defined with respect to the unit Δ-graph of the two levels. The local property therefore can be enforced in a step-by-step check by the transitions of $\mathcal{CA}$. But the local property itself is not enough to ensure that a co-T condition holds universally. A special kind of co-T ranking, called *odd* co-T ranking, is singled out. A co-T ranking is *odd* if and only if every path visits certain vertices (called *odd* vertices) infinitely many times. This *global* property can be captured by a Büchi condition, using the Miyano-Hayashi breakpoint technique for universality (alternation) elimination [16].

Let $\mathcal{A} = \langle Q, Q_0, \Sigma, \Delta, \mathcal{F}^T \rangle$ be a source T-automaton. The complementation algorithm produces a target Büchi automaton $\mathcal{CA} = \langle Q', Q'_0, \Sigma, \Delta', \langle F' \rangle \rangle$. The state set Q' is $2^Q \times 2^Q \times \mathcal{R}$, where for $\langle S, O, g \rangle \in Q'$, S records the reachable states, $O \subseteq S$ records the reachable states that have an obligation to visit *odd* vertices in the future, and g is a guessed co-T level ranking, all at the current level. The transition function $\Delta' : Q' \to 2^{Q'}$ is defined such that $\Delta'(\langle S, O, g \rangle)$ is

$$\{\ \langle \Delta(S,\sigma), \Delta(O,\sigma) \setminus odd(g'), g' \rangle : g' \in Succ(g, S, \sigma)\ \} \qquad (O \neq \emptyset), \tag{1}$$

$$\{\ \langle \Delta(S,\sigma), \Delta(S,\sigma) \setminus odd(g'), g' \rangle : g' \in Succ(g, S, \sigma)\ \} \qquad (O = \emptyset), \tag{2}$$

where $Succ(g, S, \sigma)$ returns the set of legitimate level rankings provided that the current level ranking is g and the current letter is σ, and $odd(g')$ gives the set of odd vertices at the level ranked by g'. When $\mathcal{CA}$ reads the letter $w(i)$ at level i with a level ranking f_i

and reachable state set S_i, it nondeterministically guesses a level ranking f_{i+1} such that $f_{i+1} \in Succ(f_i, S_i, w(i))$. The evolvement of both S and O are done by the classic subset construction [20] with the exception that odd vertices are excluded from O (see "$\setminus odd(g')$" in (1) and (2)). Once O becomes empty, it takes the value of the current S in the next stage (see the second S in (2)). The final state set F' is $2^Q \times \{\emptyset\} \times \mathcal{R}$. This Büchi condition $\langle F' \rangle$ requires that O be cleared infinitely often, which in turn enforces that every path visits odd vertices infinitely many times [16]. It is now clear that *Succ* represents the local property (being co-T) and $\mathcal{F}'$ captures the global property (being odd).

▶ Procedure 1 (Generic Complementation).
Input: T-automaton $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, \mathcal{F}^T \rangle$.
Output: Büchi automaton $\mathcal{CA} = \langle \Sigma, Q', Q'_0, \Delta', \langle F' \rangle \rangle$:

$$Q' = 2^Q \times 2^Q \times \mathcal{R}, \qquad Q'_0 = Q_0 \times \{\emptyset\} \times \mathcal{R},$$
$$\Delta' : Q' \to 2^{Q'} \text{ defined as in (1) and (2)}, \qquad F' = 2^Q \times \{\emptyset\} \times \mathcal{R}.$$

The state complexity of complementation is $|Q'|$. For every instantiation shown below, $|\mathcal{R}|$ dominates $2^{|Q|}$ and hence the complexity is $O(|\mathcal{R}|)$.

We now show complementation constructions for Büchi, GB and Streett, with the corresponding co-T rankings being co-Büchi, GC and Rabin, respectively. We use $\mathcal{D}^T$ to denote the set of T-ranks, $\mathcal{R}^T$ the set of T level rankings and $\mathcal{L}^T$ the set of level ranks. Clearly, $|\mathcal{R}^T| = |\mathcal{L}^T|$.

Büchi Complementation

Let $\mathcal{A} = \langle Q, Q_0, \Sigma, \Delta, \langle F \rangle \rangle$ be a Büchi automaton. $\mathscr{G}_w$ is co-Büchi accepting if *every* path in $\mathscr{G}_w$ visits F-vertices finitely often. Let $\mathcal{D}^{\mathrm{CB}}$ (the set of co-Büchi ranks) be $[2n+1]$.

▶ **Definition 1** (Co-Büchi Ranking). A co-Büchi ranking on $\mathscr{G}_w$ is a function $f : V \to \mathcal{D}^{\mathrm{CB}}$ such that:

1.1 for all vertices $v \in V$, if $f(v) \in [2n]^{\mathrm{odd}}$, then $v \notin F$;
1.2 for all edges $\langle v, v' \rangle \in E$, $f(v) \geq f(v')$.

A vertex $v \in V$ is *odd* if $f(v) \in [2n]^{\mathrm{odd}}$. A co-Büchi ranking f is *odd* if every path in $\mathscr{G}_w$ visits infinitely many odd vertices. A path ϱ stabilizes at a rank r if $(\exists i \in \mathbb{N})(\forall j \geq i), f(\varrho(j)) = f(\varrho(i)) = r$ and the smallest such i is called the *stabilization point* of ϱ. If $\mathscr{G}_w$ admits an odd co-Büchi ranking f, then by (1.2) every path eventually stabilizes at an odd rank. Then by (1.1), every path eventually does not visit F-vertices; that is, $\mathscr{G}_w$ is co-Büchi accepting.

Conversely, if $\mathscr{G}_w$ is co-Büchi accepting, then an odd co-Büchi ranking can be constructed through a series of graph transformations. Let $\mathscr{G}_0 = \mathscr{G}_w$. Vertices with only a finite number of descendants are called *finite*. Vertices that are not F-vertices and have no F-vertices as their descendants are called *F-free*. At stage 0, we assign all finite vertices rank 0 and remove them, obtaining $\mathscr{G}_1$, in which there is no finite vertices. Because $\mathscr{G}_0$ is co-Büchi accepting, there must exist in $\mathscr{G}_1$ an F-free vertex; otherwise we can select a path on which F-vertices occur infinitely often. We assign all F-free vertices rank 1 and remove them too, obtaining $\mathscr{G}_2$. Now some vertices in $\mathscr{G}_2$ are finite due to the removal of F-free vertices in stage 0. We repeat this process in the following manner: at the first phase of stage i, we assign even rank $2i$ to finite vertices and remove them; at the second phase, we assign F-free vertices odd rank $2i+1$ and remove them. By F-freeness, removing F-free vertices from $\mathscr{G}_{2i+1}$ gets rid of at least one infinite path, and hence $width(\mathscr{G}_{2i+2}) < width(\mathscr{G}_{2i})$. Therefore, this process

terminates at a stage $j \leq n$. In the following summary, by $\mathscr{G} \setminus V$ we mean removing from $\mathscr{G}$ all vertices in V and their incoming and outgoing edges.

▶ Procedure 2 (Co-Büchi Ranking Assignment).
Input: a co-Büchi accepting $\mathscr{G}_0$. Output: a co-Büchi ranking f. Repeat for $i \in [0..n]$ if $\mathscr{G}_{2i} \neq \emptyset$.

2.1 (a) $V_{2i} = \{v \in V \mid v \textit{ is finite in } \mathscr{G}_{2i}\}$;
(b) $f(v) = 2i$ for $v \in V_{2i}$;
(c) $\mathscr{G}_{2i+1} = \mathscr{G}_{2i} \setminus V_{2i}$.

2.2 (a) $V_{2i+1} = \{v \in V \mid v \textit{ is } F\textit{-free in } \mathscr{G}_{2i+1}\}$;
(b) $f(v) = 2i+1$ for $v \in V_{2i+1}$;
(c) $\mathscr{G}_{2i+2} = \mathscr{G}_{2i+1} \setminus V_{2i+1}$.

▶ **Lemma 2** ([11]). *$\mathscr{G}_w$ is co-Büchi accepting if and only if $\mathscr{G}_w$ admits an odd co-Büchi ranking.*

We have $|\mathcal{D}^{\mathrm{CB}}| = O(n)$, and hence $|\mathcal{R}^{\mathrm{CB}}| = (O(n))^n = 2^{O(n \lg n)}$.

GB Complementation

Let $\mathcal{A} = \langle Q, Q_0, \Sigma, \Delta, \langle B \rangle_I \rangle$ be a generalized Büchi automaton. GC ranking is meant to be used for GB complementation. A $\mathscr{G}_w$ is GC accepting if for *every* path ϱ in $\mathscr{G}_w$ there exists $j \in I$ such that ϱ only visits $B(j)$-vertices finitely often. Let $\mathcal{D}^{\mathrm{GC}} = ([2n]^{\mathrm{odd}} \times I) \cup [2n+1]^{\mathrm{even}}$ be the set of GC ranks. We refer to values in $[2n]^{\mathrm{odd}} \times I$ as *odd* ranks, and values in $[2n+1]^{\mathrm{even}}$ as *even* ranks. For an odd GC rank $\langle t, u \rangle$, we call t numeric rank (r-rank) and u index rank (h-rank). Even GC ranks are just numeric ranks. The greater-than and less-than orders on GC ranks are solely defined on r-ranks. For example, $\langle t, u \rangle > \langle t', u' \rangle$ (or $\langle t, u \rangle > t'$, $t > \langle t', u' \rangle$) if and only if $t > t'$. This definition is sound with respect to its usage in this paper; as shown below, we never need to compare two odd GC ranks having the same r-rank but different h-ranks.

▶ **Definition 3** (GC Ranking). A GC ranking on $\mathscr{G}_w$ is a function $f : V \to \mathcal{D}^{GC}$ such that:

3.1 for every vertex $v \in V$, if $f(v) = \langle 2i+1, j \rangle$ for some $j \in I$, then $v \notin B(j)$;
3.2 for every edge $\langle v, v' \rangle \in E$, $f(v) \geq f(v')$.

A vertex v is called *odd* (resp. *even*) if $f(v) \in [2n]^{\mathrm{odd}} \times I$ (resp. $f(v) \in [2n+1]^{\mathrm{even}}$). A GC ranking f is *odd* if every path in $\mathscr{G}_w$ visits infinitely many odd vertices. Note that (3.2) implies that if two adjacent odd vertices have the same r-rank, then they have the same h-rank. As in Büchi complementation, if $\mathscr{G}_w$ admits an odd GC ranking, then every path eventually stabilizes at an odd GC rank $\langle t, j \rangle$, and from the stabilization point on never visits $B(j)$-vertices. Therefore, $\mathscr{G}_w$ is GC accepting.

Conversely, if $\mathscr{G}_w$ is GC accepting, then we can find a GC ranking by a series of graph transformations as in Büchi complementation. Each stage has two phases. We begin stage i with $\mathscr{G}_{2i}$ ($\mathscr{G}_0 = \mathscr{G}_w$). In the first phase, finite vertices receive even rank $2i$ and are removed, resulting in $\mathscr{G}_{2i+1}$. Thanks to GC condition, if $\mathscr{G}_{2i+1}$ is not empty, then for some $j \in I$ and $v \in V$, v is $B(j)$-free. In the second phase, those $B(j)$-free vertices receive odd rank $\langle 2i+1, j \rangle$ and are removed, producing $\mathscr{G}_{2i+2}$. This procedure repeats for all $i \in [0..n]$ unless $\mathscr{G}_{2i}$ is empty. The termination condition is justified by $\mathit{width}(\mathscr{G}_{2i+2}) < \mathit{width}(\mathscr{G}_{2i})$, just as before.

▶ Procedure 3 (GC Ranking Assignment).
Input: a GC accepting $\mathscr{G}_0$. Output: a GC ranking f. Repeat for $i \in [0..n]$ if $\mathscr{G}_{2i} \neq \emptyset$.

3.1 (a) $V_{2i} = \{v \in V \mid v \textit{ is finite in } \mathscr{G}_{2i}\}$;
(b) $f(v) = 2i$ for $v \in V_{2i}$;
(c) $\mathscr{G}_{2i+1} = \mathscr{G}_{2i} \setminus V_{2i}$.

3.2 (a) $V_{2i+1} = \{v \in V \mid v$ is $B(j)$-free in $\mathscr{G}_{2i+1}\}$ for a $j \in I$ such that $B(j)$-free vertices exist;
(b) $f(v) = \langle 2i+1, j\rangle$ for $v \in V_{2i+1}$;
(c) $\mathscr{G}_{2i+2} = \mathscr{G}_{2i+1} \setminus V_{2i+1}$.

In (3.2), it does not matter which $j \in I$ is chosen. But this flexibility plays an important role in our Streett complementation construction (see Procedure 5).

▶ **Lemma 4** ([12]). *$\mathscr{G}_w$ is* GC *accepting if and only if $\mathscr{G}_w$ admits an odd* GC *ranking.*

We have $|\mathcal{D}^{\mathrm{GC}}| = O(nk)$, and hence $|\mathcal{R}^{\mathrm{GC}}| = (O(nk))^n = 2^{O(n \lg nk)}$.

Streett Complementation

Let $\mathcal{A} = \langle Q, Q_0, \Sigma, \Delta, \langle G, B\rangle_I\rangle$ be a Streett automaton. Rabin ranking is meant for Streett complementation. Let us first examine the simple case where $k = 1$, i.e., every path satisfies $[G(1), B(1)]$. Easily seen, $\mathscr{G}_w$ admits a co-Büchi ranking, and hence we can instantiate Procedure 1 with $\mathcal{R}$ being co-Büchi level rankings (which are also GC level rankings with index size 1). The only modification needed is to enforce that every path visits $G(1)$-vertices, which can be easily realized by a Büchi accepting condition (see the definition of $\langle F'\rangle$ in Procedure 1). This simple procedure fails for $k > 1$, because a path visiting a finite number of $B(j)$-vertices may not have to visit infinitely many $G(j)$-vertices; it just satisfies $[G(j'), B(j')]$ for $j' \neq j$. Nevertheless, if we could find a way to reduce the number of Rabin pairs one by one, eventually the simple scenario has to occur. The idea in [14] is to use GC rankings to approximate Rabin accepting behaviors step by step until finally obtaining the precise characterization. As a result, Rabin ranks are tuples of GC ranks, considerably more sophisticated than GC ranks. We first put aside the formal definition of Rabin rankings and show how a Rabin ranking can be obtained provided $\mathscr{G}_w$ is Rabin accepting. Once again, this is done through a series of graph transformations.

▶ Procedure 4 (Rabin Ranking Assignment).
Input: a Rabin accepting $\mathscr{G}_0$. Output: a Rabin ranking f. Repeat for $i \in [0..k]$ if $\mathscr{G}_i \neq \emptyset$.

4.1 Assign $\mathscr{G}_i$ a GC ranking gc_{i+1}.
4.2 Remove all vertices v if $\mathrm{gc}_{i+1}(v)$ is even.
4.3 Remove all edges $\langle v, v'\rangle$ if $\mathrm{gc}_{i+1}(v) > \mathrm{gc}_{i+1}(v')$.
4.4 Remove all edges $\langle v, v'\rangle$ if $\mathrm{gc}_{i+1}(v)$ is odd with index j and v is a $G(j)$-vertex.
4.5 $f(v) = \langle \mathrm{gc}_1(v), \ldots \mathrm{gc}_{i+1}(v)\rangle$ iff v is removed from $\mathscr{G}_i$.

Obviously, if $\mathscr{G}_w$ is Rabin accepting for a Rabin condition $[G, B]_I$, then it is also GC accepting for the GC condition $[B]_I$. By Lemma 4, a GC ranking gc_1 exists for $\mathscr{G}_0$, which justifies Step (4.1) at stage 0. Steps (4.2)-(4.3) may break up $\mathscr{G}_0$ into a collection of graph components (in the undirected sense). Let $\mathcal{C}$ be such a component. Steps (4.2)-(4.3) ensure that vertices in $\mathcal{C}$ have the same odd rank with some index $j \in I$, and hence all are $B(j)$-free. Step (4.4), deleting all outgoing edges from $G(j)$-vertices, may further break up $\mathcal{C}$ into more components. In particular, any infinite path is destroyed (i.e., broken into a collection of finite paths) if the path satisfies $[G(j), B(j)]$ (i.e., visiting infinitely many $G(j)$-vertices but only finitely many $B(j)$-vertices). Let $\mathcal{C}' \subseteq \mathcal{C}$ be a resulting component after Step (4.4). As a result, $\mathcal{C}'$ should satisfy the *reduced* Rabin condition $[G, B]_{I\setminus\{j\}}$, and hence the *reduced* GC condition $[B]_{I\setminus\{j\}}$.

So after stage 0, $\mathscr{G}_1$ is composed of a collection of pairwise disjoint components, each of which satisfies a Rabin condition whose cardinality is at most $k-1$. Precisely speaking, at the beginning of each stage $i \geq 1$, $\mathscr{G}_i$ is composed of a collection of pairwise disjoint components, and at Step (4.1), gc_{i+1} is obtained by *independently* assigning each component in $\mathscr{G}_i$ a GC ranking according to the *reduced* GC condition the component satisfies. By induction, at stage $i \geq 1$, vertices in each component in $\mathscr{G}_i$ have been assigned the same tuple of odd GC ranks of length i and each component satisfies a Rabin condition whose cardinality is at most $k-i$. It follows that the procedure terminates and each vertex in $\mathscr{G}_w$ eventually gets a tuple of GC ranks of length at most $k+1$. Note that the last GC rank in a tuple is always an even GC rank (r-rank).

Let $\mathcal{D}^{\mathrm{R}}$ denote the set of Rabin ranks of the form $\langle\langle r_1, i_1\rangle, \ldots, \langle r_m, i_m\rangle, r_{m+1}\rangle$ $(m \leq k)$. Ordering relations $(<_m, \leq_m, >_m, \geq_m, =_m)$ on Rabin ranks are defined to be the standard lexicographical extension (up to m-th component) of orderings on GC ranks. For a Rabin rank γ of the above form, the *index projection* (or the *h-projection*) of γ, written $\mathrm{Proj}_{\mathrm{h}}\,\gamma$, is $\langle i_1, \ldots, i_m\rangle$ and the *numeric projection* (or the *r-projection*) of γ, written $\mathrm{Proj}_{\mathrm{r}}\,\gamma$, is $\langle r_1, \ldots, r_{m+1}\rangle$. With respect to a given function $f : V \to \mathcal{D}^{\mathrm{R}}$, the *width* of $v \in V$ is the length of $f(v)$, denoted by $|v|_f$ (or $|v|$, when f is clear from the context). We say that v is *odd* (called *happy* in [14]) if $|v| > 1$ and v is a $G(|v|-1)$-vertex. We arrive at the formal definition of Rabin rankings.

▶ **Definition 5** (Rabin Ranking). A Rabin ranking is a function $f : V \to \mathcal{D}^{\mathrm{R}}$ satisfying the following conditions.

5.1 For every vertex $v \in V$ with $|v| = m+1 \geq 2$ and $\alpha = \mathrm{Proj}_{\mathrm{h}}\, f(v)$, we have
- **a.** for $i \in [1..m)$, $v \notin G(\alpha[i])$;
- **b.** for $i \in [1..m]$, $v \notin B(\alpha[i])$.

5.2 For every edge $\langle v, v'\rangle \in E$ with $|v| = m+1$, $|v'| = m'+1$ and $m'' = \min(m, m')$, we have
- **a.** $f(v) \geq_{m''} f(v')$;
- **b.** $f(v) \geq_{m''+1} f(v')$, or v is odd.

A Rabin ranking is *odd* if every path in $\mathscr{G}_w$ visits infinitely many odd vertices.

▶ **Lemma 6** ([14]). *$\mathscr{G}_w$ is Rabin accepting if and only if $\mathscr{G}_w$ admits an odd Rabin ranking.*

We have $|\mathcal{D}^{\mathrm{R}}| = (O(nk))^{k+1}$ and hence $|\mathcal{R}^{\mathrm{R}}| = ((O(nk))^{k+1})^n = (nk)^{O(nk)} = 2^{O(nk \lg nk)}$.

4 Improved Streett Complementation

The above construction requires $2^{O(nk \lg nk)}$ state blow-up [14], which is substantially larger than the lower bound in [3]. In the extreme case of $k = O(2^n)$, the construction is double exponential in n. Intuitively, the larger the k, the more overlaps between $B(i)$'s and between $G(i)$'s $(i \in I)$. A natural question is: can all Rabin pairs $[G(i), B(i)]$ independently impose behaviors on a Rabin accepting $\mathscr{G}_w$? We showed in [2] that in Rabin complementation we can build a Streett accepting $\mathscr{G}_w$ for which no Streett pair $\langle G(i), B(i)\rangle$ is redundant. We observed the opposite in Streett complementation; the larger the k, the higher the correlation between infinite paths that satisfy $[G, B]_I$. By exploiting this correlation, we can walk in *big* steps in approximating Rabin accepting behaviors using GC rankings. As a result, our Rabin ranks are tuples of GC ranks of length at most $\mu = \min(n, k)$. This simple but crucial observation leads to a significant improvement on the construction complexity. We elaborate on this below.

The first idea is that at Step (4.4) in stage i, in a component $\mathcal{C}$ of $\mathscr{G}_i$, instead of removing all outgoing edges from $G(j)$-vertices, we can remove all outgoing edges from $G(j')$-vertices

for all j' such that $B(j') \subseteq B(j)$. Let $J = \{j' \in I \mid B(j') \subseteq B(j)\}$. Since vertices in $\mathcal{C}$ are $B(j)$-free, they are also $B(j')$-free for any $j' \in J$. Recall that Step (4.4) is to break all infinite paths that satisfy $[G(j), B(j)]$ so that in each resulting component we have a simpler Rabin condition to satisfy. If an infinite path in $\mathcal{C}$ satisfies $[G, B]_J$, then removing all outgoing edges from $G(j')$-vertices (for $j' \in J$) certainly serves the same purpose, and moreover, any resulting component only needs to satisfy a Rabin condition whose cardinality is $|J|$ less than at the beginning of stage i.

The second idea is that at Step (4.1) in stage i, we can assign special GC rankings to components in $\mathscr{G}_i$. Recall that a GC ranking is obtained by a series of graph transformations too. In Step (3.2), we assign and remove $B(j)$-free vertices for some $j \in I$. In fact any fixed $j \in I$ is sufficient as long as $B(j)$-free vertices exist. Therefore, we can choose a j such that not only $B(j)$-free vertices exist, but also for any other $j' \in I$, $B(j') \not\subset B(j)$, if $B(j')$-free vertices also exist. Intuitively, we prefer a j such that $B(j)$ is minimal (with respect to set inclusion) because more vertices would be $B(j)$-free and subject to removal.

We refine those ideas by taking into account the history of GC rankings. In stage i, right before Step (4.1), vertices in $\mathscr{G}_i$ were assigned a tuple of GC ranks of length i. Consider a component $\mathcal{C} \subseteq \mathscr{G}_i$. No vertices in $\mathcal{C}$ received an even GC rank in stage $i' \in [0..i)$, because otherwise they were already removed by Step (4.2) in that stage. Also all vertices in $\mathscr{G}_i$ received the same odd GC rank in each stage $i' \in [0..i)$, for otherwise Step (4.3) in stage i' would have broken the component. Now let $\langle\langle r_1, j_1\rangle, \ldots, \langle r_i, j_i\rangle\rangle$ be the tuple that has been assigned to all vertices in $\mathcal{C}$. Let $B' = \cup_{t\in[1..i]} B(j_t)$ and $J' = \{j' \in I \mid B(j') \subseteq B'\}$. So all vertices in $\mathcal{C}$ are B'-free. When we assign GC rankings for $\mathcal{C}$, in each stage at Step (3.2) (in Procedure 3), we choose a $j \in I \setminus J'$ such that (1) $B(j)$-free vertices exist, (2) $B(j) \not\subseteq B'$ (we say $B(j)$ is not *covered* by B'), and (3) no $B(j')$-free vertices exist for any other $j' \in I \setminus J'$ with $B' \cup B(j') \subset B' \cup B(j)$. In other words, we choose a j such that not only we can find $B(j)$-free vertices, but also $B' \cup B(j)$ minimally extends B'. Now let $\mathcal{C}'$ be a component right before Step (4.4) is taken and let $\langle\langle r_1, j_1\rangle, \ldots, \langle r_{i+1}, j_{i+1}\rangle\rangle$ be the tuple of ranks that has been assigned to all vertices in $\mathcal{C}'$ (for the same reason as before, all vertices in $\mathcal{C}'$ have received the same sequence of odd GC ranks). Let $B'' = B' \cup B(j_{i+1})$ and $J'' = \{j'' \in I \mid B(j'') \subseteq B''\}$. In Step (4.4) we remove all outgoing edges of $G(j'')$-vertices if $B(j'') \subseteq B''$. This deletion destroys all infinite paths that satisfy $[G, B]_{J''}$ because all vertices in $\mathcal{C}'$ are B''-free. Let $\mathcal{C}''$ be a resulting component and ϱ an infinite path in $\mathcal{C}''$. Then ϱ only needs to satisfy $[G, B]_{I\setminus J''}$.

Now let us assume that we have incorporated the above ideas into Procedure 4 and obtained a new Rabin rank $\gamma = \langle\langle r_1, j_1\rangle, \ldots, \langle r_m, j_m\rangle, r_{m+1}\rangle\rangle$. Let $\alpha = \mathrm{Proj_h}\,\gamma$. The *non-covering* condition stated above requires α to satisfy:

$$\forall i \in [1..m]\ B(\alpha[i]) \not\subseteq \cup_{j=1}^{i-1} B(\alpha[j]), \tag{3}$$

which implies $|\alpha| \leq n$. By definition, $|\alpha| \leq k$ and so we have $|\alpha| \leq \mu$ and $|\gamma| \leq \mu + 1$. From now on we switch to terms μR *ranks* (i.e., *minimal Rabin ranks*), μR *rankings*, μR *level rankings* and μR *level ranks*. Their precise definitions are to be given below. We define two functions $\mathrm{Cover} : I^* \to 2^I$ and $\mathrm{Mini} : I^* \to 2^I$ to formalize the intuition of *minimal extension*. Cover maps tuples of indices to subsets of I such that

$$\mathrm{Cover}(\alpha) = \{\, j \in I \ \mid\ B(j) \subseteq \cup_{i=1}^{|\alpha|} B(\alpha[i]) \,\}.$$

Note that $\mathrm{Cover}(\epsilon) = \emptyset$. Mini maps tuples of indices to subsets of I such that $j \in \mathrm{Mini}(\alpha)$ if and only if $j \in I \setminus \mathrm{Cover}(\alpha)$ and

$$\forall j' \in I \setminus \mathrm{Cover}(\alpha)\ \big(j' \neq j \to B(j') \cup \mathrm{Cover}(\alpha) \not\subset B(j) \cup \mathrm{Cover}(\alpha)\big), \tag{4}$$

$$\forall j' \in I \setminus \mathrm{Cover}(\alpha)\left(j' < j \to B(j') \cup \mathrm{Cover}(\alpha) \neq B(j) \cup \mathrm{Cover}(\alpha)\right). \tag{5}$$

$\mathrm{Mini}(\alpha)$ consists of choices of indices to *minimally* enlarge $\mathrm{Cover}(\alpha)$; ties (with respect to set inclusion) are broken by numeric minimality (Condition (5)). Before introducing μR ranking assignment, we need a new GC ranking assignment which takes a tuple of I-indices as an additional input. We call so obtained GC rankings (resp. GC ranks) μGC rankings (resp. μGC ranks).

▶ Procedure 5 (μGC Ranking Assignment).
Input: a GC accepting $\mathscr{G}_0$, a tuple of I-indices α. Output: a μGC ranking f.
Repeat for $i \in [0..n]$ if $\mathscr{G}_{2i} \neq \emptyset$.

5.1 (a) $V_{2i} = \{v \in V \mid v \textit{ is finite in } \mathscr{G}_{2i}\}$;
(b) $f(v) = 2i$ for $v \in V_{2i}$;
(c) $\mathscr{G}_{2i+1} = \mathscr{G}_{2i} \setminus V_{2i}$.

5.2 (a) $V_{2i+1} = \{v \in V \mid v \text{ is } B(j)\text{-free in } \mathscr{G}_{2i+1}\}$ for a $j \in \mathrm{Mini}(\alpha)$ such that $B(j)$-free vertices exist;
(b) $f(v) = \langle 2i+1, j\rangle$ for $v \in V_{2i+1}$;
(c) $\mathscr{G}_{2i+2} = \mathscr{G}_{2i+1} \setminus V_{2i+1}$.

As in GC ranking assignment, in (5.2) there maybe more than one j such that $B(j)$-free vertices exist, and it does not matter which one we choose. But in case $\mathrm{Mini}(\alpha)$ is a singleton, we have a unique j at all stages, essentially synchronizing all h-ranks in the μGC ranks obtained. This synchronization is crucial in our construction for parity complementation (see Section 6).

▶ Procedure 6 (μR Ranking Assignment).
Input: a Rabin accepting $\mathscr{G}_0$. Output: a μR ranking f. Repeat for $i \in [0..\mu]$ if $\mathscr{G}_i \neq \emptyset$.

6.1 Assign $\mathscr{G}_i$ a μGC ranking gc_{i+1}.
6.2 Remove all vertices $v \in V$ if $\mathrm{gc}_{i+1}(v)$ is even.
6.3 Remove all edges $\langle v, v'\rangle \in E$ if $\mathrm{gc}_{i+1}(v) > \mathrm{gc}_{i+1}(v')$.
6.4 Remove all edges $\langle v, v'\rangle \in E$ if $v \in G(t)$ for some $t \in \mathrm{Cover}(\mathrm{Proj}_{\mathrm{h}}(\langle \mathrm{gc}_1, \ldots, \mathrm{gc}_{i+1}\rangle))$.
6.5 $f(v) = \langle \mathrm{gc}_1(v), \ldots \mathrm{gc}_{i+1}(v)\rangle$ iff v is removed from $\mathscr{G}_i$.

Note that Step (6.1) actually means that Procedure 5 is called upon for every component $\mathcal{C} \subset \mathscr{G}_i$, with the corresponding α being $\mathrm{Proj}_{\mathrm{h}}(\langle \mathrm{gc}_1(v), \ldots, \mathrm{gc}_i(v)\rangle)$ for some $v \in \mathcal{C}$ (α is well-defined since all vertices in $\mathcal{C}$ have received the same sequence of μGC ranks). It is time to formally define μR ranking. Let f be a function $V \to (\mathcal{D}^{\mathrm{GC}})^{\mu+1}$. We say that v is *odd* if $|v| > 1$ and v is a $G(t)$-vertex for some $t \in \mathrm{Cover}(\alpha[1..|v|-1])$ where $\alpha = \mathrm{Proj}_{\mathrm{h}} f(v)$.

▶ **Definition 7** (μR Ranking). A μR ranking is a function $f : V \to (\mathcal{D}^{\mathrm{GC}})^{\mu+1}$ satisfying the following conditions.

7.1 For every vertex $v \in V$ with $|v| = m+1 \geq 2$ and $\alpha = \mathrm{Proj}_{\mathrm{h}} f(v)$, we have
- **a.** for $i \in [1..m)$, $v \notin G(t)$ for $t \in \mathrm{Cover}(\alpha[1..i])$;
- **b.** for $i \in [1..m]$, $v \notin B(t)$ for $t \in \mathrm{Cover}(\alpha[1..i])$;
- **c.** for $i \in [1..m]$, $\alpha[i] \in \mathrm{Mini}(\alpha[1..i))$.

7.2 For every edge $\langle v, v'\rangle \in E$ with $|v| = m+1$, $|v'| = m'+1$ and $m'' = \min(m, m')$, we have
- **a.** $f(v) \geq_{m''} f(v')$;
- **b.** $f(v) \geq_{m''+1} f(v')$, or v is odd.

A μR ranking is *odd* if every infinite path in $\mathscr{G}_w$ visits infinitely many odd vertices.

▶ **Lemma 8.** *$\mathscr{G}_w$ is Rabin accepting if and only if $\mathscr{G}_w$ admits an odd μR ranking.*

5 Complexity

In this section we analyze the complexity of our construction. As shown below, all complexity related notions X are also parameterized with B (besides n and k). Still, we choose to list some or all of them when clarity is needed. In particular, we use $|X|$ to abbreviate $|X(n,k)| = \max_B |X(B,n,k)|$.

Let $\mathcal{D}^{\mu\mathrm{R}}$ be the set of μR ranks that can be produced by Procedure 6. Formally, $\mathcal{D}^{\mu\mathrm{R}} = \cup_f \mathrm{range}(f)$ where f ranges over all possible outputs of Procedure 6. In a similar manner, we define $\mathcal{R}^{\mu\mathrm{R}}$ and $\mathcal{L}^{\mu\mathrm{R}}$. We have $|\mathcal{R}^{\mu\mathrm{R}}| = |\mathcal{L}^{\mu\mathrm{R}}|$. We note that these notions are defined differently from their counterparts in Section 3; due to two kinds of correlations (which we refer to as *horizontal* and *vertical* correlation), $|\mathcal{L}^{\mu\mathrm{R}}|$ is much smaller than $|\mathcal{D}^{\mu\mathrm{GC}}|^{n(\mu+1)}$. We view $\mathcal{L}^{\mu\mathrm{R}}$ as a set of $n \times (\mu+1)$ matrices of μGC ranks and carry out a further simplification. Let $\mathcal{M}^{\mu\mathrm{R}}$ be a set of $n \times \mu$ matrices obtained from $\mathcal{L}^{\mu\mathrm{R}}$ by the following mapping: each $n \times (\mu+1)$ matrix M is mapped to an $n \times \mu$ matrix M' by (a) deleting from M even ranks at the end of each row, (b) changing odds rank of the form $\langle 2i-1, j\rangle$ to $\langle i, j\rangle$, and (c) aligning each row to length μ by filling $\langle 1,0\rangle$'s. Clearly, $|\mathcal{L}^{\mu\mathrm{R}}| \le n^n \cdot |\mathcal{M}^{\mu\mathrm{R}}|$; the factor n^n suffices to compensate (a), the deletion of even ranks, and (b) and (c) have no effect to the cardinality (for (b), $i \to 2i-1$ is one-to-one from $[1..n]$ onto $[2n]^{\mathrm{odd}}$). Let $M \in \mathcal{M}^{\mu\mathrm{R}}$ be called μR-matrices. We write $\mathrm{Proj}_{\mathrm{r}}\, M$ and $\mathrm{Proj}_{\mathrm{h}}\, M$ to mean, respectively, the projection of M on numeric ranks (called an r-matrix) and on index ranks (called an h-matrix). Let $\mathcal{M}^r = \mathrm{Proj}_{\mathrm{r}}\, \mathcal{M}^{\mu\mathrm{R}}$, $\mathcal{M}^h = \mathrm{Proj}_{\mathrm{h}}\, \mathcal{M}^{\mu\mathrm{R}}$, and $\mathcal{D}^r$ and $\mathcal{D}^h$ be the sets of rows occurring in matrices in $\mathcal{M}^r$ and $\mathcal{M}^h$, respectively. Obviously, we have $|\mathcal{M}^{\mu\mathrm{R}}| \le |\mathcal{M}^r| \cdot |\mathcal{M}^h|$ and $|\mathcal{M}^h| \le (|\mathcal{D}^h|)^n$.

▶ **Example 9** (μR-Matrix)**.** Let us consider a case where $n=3$, $k=3$, and $Q=\{q_0,q_1,q_2\}$. Below we show that a μR level rank f corresponds to a μR-matrix M, which projects to M_r and M_h.

$$\begin{array}{ccc|cccc|ccc|ccc|ccc}
\begin{bmatrix} q_0 \\ q_1 \\ q_2 \end{bmatrix} & & & \left|\begin{matrix} \langle 1,2\rangle & \langle 1,3\rangle & 4 & \\ \langle 1,2\rangle & \langle 3,1\rangle & 2 & \\ \langle 1,2\rangle & \langle 3,1\rangle & \langle 3,3\rangle & 0 \end{matrix}\right| & & & & \left|\begin{matrix} \langle 1,2\rangle & \langle 1,3\rangle & \langle 1,0\rangle \\ \langle 1,2\rangle & \langle 2,1\rangle & \langle 1,0\rangle \\ \langle 1,2\rangle & \langle 2,1\rangle & \langle 2,3\rangle \end{matrix}\right| & & & \left|\begin{matrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 2 & 2 \end{matrix}\right| & & & \left|\begin{matrix} 2 & 3 & 0 \\ 2 & 1 & 0 \\ 2 & 1 & 3 \end{matrix}\right| \\
Q & & & f & & & & M & & & M_r & & & M_h & &
\end{array}$$

Bounding $|\mathcal{M}^h|$

It turns out that we only need to exploit a horizontal correlation to bound $|\mathcal{M}^h|$. Recall that each $\alpha \in I^*$ names a subset of Q, namely $\mathrm{Cover}(\alpha)$. The idea is to order all α that could occur in $\mathcal{D}^h$ into a tree structure. Consider an unordered tree where the root is labeled by ϵ and each non-root node is labeled by an index in I. With little confusion, we identify a node α with the path from the root to α and represent α by the sequence of indices on the path. So a non-root node α has $\alpha[|\alpha|]$ as its label and names $\mathrm{Cover}(\alpha)$. We arrive at the following important notion.

▶ **Definition 10** (Increasing Tree of Sets (ITS))**.** An ITS $T(n,k,B)$ is an unordered I-labeled tree (except the root which is labeled by ϵ) such that

10.1 A non-root node α exists in $T(n,k,B)$ iff $\forall i \in [1..|\alpha|]$, $\alpha[i] \in \mathrm{Mini}(\alpha[1..i])$.

Property (10.1) succinctly encodes three important features of ITS. First, an ITS is maximal in the sense that no node can be added. Second, if β is a direct child of α, then β must name at least one new state that has not been named by α. Third, the new contributions

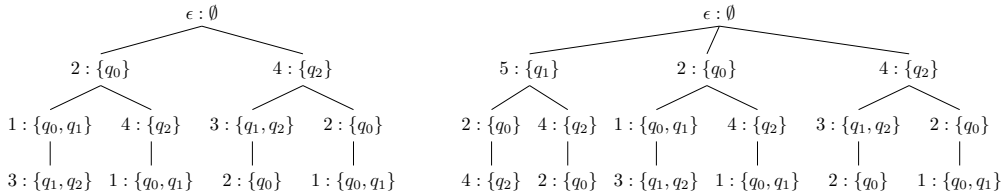


Figure 2 Two ITS in Example 11.

by β cannot be covered by contributions made by any another sibling β'. In particular, if more than one sibling can make the same contribution, then the one with the smallest index is selected. It follows that each tuple of n, k and B uniquely determines $T(n, k, B)$ (in the unordered sense). Note that the height of $T(n, k, B)$ (the length of the longest path in $T(n, k, B)$) is bounded by μ.

▶ **Example 11** (ITS)**.** Consider $n = 3$, $k = 4$, $Q = \{q_0, q_1, q_2\}$, $B : [1..4] \to 2^Q$ and $B' : [1..5] \to 2^Q$,

$$B(1) = \{q_0, q_1\}, \qquad B(2) = \{q_0\}, \qquad B(3) = \{q_1, q_2\}, \qquad B(4) = \{q_2\},$$

and B' extends B with $B'(5) = \{q_1\}$. $T(3, 4, B)$ and $T(3, 5, B')$ are given in Figure 2. For clarity, for each non-root node α, we also list $B(\alpha[|\alpha|])$ as the set label of α. In $T(3, 4, B)$, neither $\{q_0, q_1\}$ nor $\{q_1, q_2\}$ can appear at height 1, because $\{q_0, q_1\}$ covers $\{q_0\}$ and $\{q_1, q_2\}$ covers $\{q_2\}$. The leftmost node at the bottom level is labeled by $\{q_1, q_2\}$ instead of by $\{q_2\}$ due to the index minimality requirement. For the same reason, in $T(3, 5, B')$, we have nodes $\langle 2, 1\rangle$, $\langle 2, 4, 1\rangle$ and $\langle 4, 2, 1\rangle$ all labeled with $\{q_0, q_1\}$, and nodes $\langle 2, 1, 3\rangle$ and $\langle 4, 3\rangle$ all labeled with $\{q_1, q_2\}$.

It is easily seen that Property (7.1c) corresponds exactly to Property (10.1). So a one-to-one correspondence exists between non-root nodes in $T(n, k, B)$ and elements in $\mathcal{D}^h(n, k, B)$. Let $|T(n, k, B)|$ denote the number of non-root nodes in $T(n, k, B)$ and $H(n, k) = \max_B |T(n, k, B)|$. Clearly, we have $|\mathcal{M}^h| \leq (H(n, k))^n$.

▶ **Lemma 12.** *$H(n, k) = 2^{O(k \lg k)}$ for $k = O(n)$ and $H(n, k) = 2^{O(n \lg n)}$ for $k = \omega(n)$.*

Bounding $|\mathcal{M}^r|$

Here we need to exploit both horizontal and vertical correlations. We show that every $n \times \mu$ r-matrix induces a 2^Q-labeled ordered tree with at most n leaves and with height at most μ. Such a tree is called an $n \times \mu$ tree. We bound $|\mathcal{M}^r|$ by counting the number of $n \times \mu$ trees.

Let M be an r-matrix. Since M comes from a μR level rank, M is associated with vertices at a level. To facilitate the discussion below, we use term *states* to specifically mean those vertices at the level where M is associated with, and use term *vertices* just as before. By rank i we simply mean a number i in M, which corresponds to the numeric μGC rank $2i - 1$.

Let us first consider ranks in column 1 of M. A state q being ranked with an odd μGC rank means that at certain stage of μGC ranking assignment, q becomes $B(j)$-free for some $j \in I$, which implies that there exists an infinite path starting from q (recall that all finite

vertices have been removed before this odd μGC rank is assigned). If two states q, q' are ranked with different odd μGC ranks, say q with $\langle 2i-1, j\rangle$ and q' with $\langle 2i'-1, j'\rangle$ where $i > i'$, then there exist two infinite paths ϱ and ϱ' such that ϱ starts from q, ϱ' starts from q', and ϱ and ϱ' never intersect. This is due to the nature of μGC ranking assignment; $\langle 2i-1, j\rangle$ is assigned to some $B(j)$-free vertices only after those $B(j')$-free vertices with odd μGC rank $\langle 2i'-1, j'\rangle$ have been removed.

Note that it is perfectly possible that an infinite path starting from q intersects another infinite path starting from q'. But a maximal subset $S^{(1)}$ of states, all with the same rank, called a *cell* at column 1, should "own" at least one *private* infinite path that does not intersect the private paths owned by any other cells at column 1. We call a path *named* if it is owned by a cell. Let $m^{(1)}$ be the maximum rank in column 1, and note that not all ranks in $[1..m^{(1)}]$ necessarily appear in column 1. But again, by the way μGC ranking assignment is carried out, for each non-occurring rank, at least one private infinite path exists, which is called *hidden* and viewed as being owned by $\emptyset$. Easily seen now, each rank in $[1..m^{(1)}]$ corresponds to a non-empty set of private infinite paths.

In general, a *cell* at column l is a maximal subset of states, each of which is assigned the same tuple of ranks up to column l. Consider a cell $S^{(l)} = \{q_{i_1}, \ldots, q_{i_j}\}$ at column l. Let $m^{(l+1)} = \max\{M[i_1, l+1], \ldots, M[i_j, l+1]\}$. By the same reasoning as before, each rank in $[1..m^{(l+1)}]$ corresponds to a non-empty set of private infinite paths. The private paths associated with rank $M[i_{j'}, l+1]$ are owned by $S^{(l+1)}_{j'} \subseteq S^{(l)}$ which is a cell at column $l+1$ with rank $M[i_{j'}, l+1]$ ($S^{(l+1)}_{j'} = \emptyset$ if no states in $S^{(l)}$ is mapped to $M[i_{j'}, l+1]$). Moreover, none of these paths, hidden or named, should intersect private paths owned by any cell at column l that is a subset of $Q \setminus S^{(l)}$, because states in $Q \setminus S^{(l)}$ and states in $S^{(l)}$ are not in the same component at stage l (the $l+1$-th stage) in Procedure 6. Now we are ready to show how to build an $n \times \mu$ tree from M.

Each node in the tree is associated with a label which is a subset of Q. The root is labeled with set Q. For each rank $i \in [1..m^{(1)}]$, we add a child to the root and we order those children increasingly by the ranks associated with them. If i does not appear in column 1, the i-th child (from left to right) is labeled with $\emptyset$ and is a terminal node (leaf). Otherwise, the child is labeled with the cell at column 1 with rank i and the child is non-terminal if its height is less than μ. We repeat the process column by column. Each maximal $S^{(l)}$ at column $l < \mu$ corresponds to a non-terminal node at height l, from which we spawn a child for each rank in $i \in [1..m^{(l)}]$, and we order and label the children using the rule stated above. After processing column μ, we obtain an $n \times \mu$ tree, because the number of leaves in the tree cannot exceed $\mathit{width}(\mathscr{G}_w) \leq n$, which we refer to as the *maximum width property* (MWP). Now we call an $n \times \mu$ tree a TOP (*Tree of Ordered Partitions*) and let $\mathcal{T}^r(n,k)$ denote the set of TOPs. We have $|\mathcal{M}^r| \leq |\mathcal{T}^r(n,k)|$.

▶ **Example 13** (TOP). Four 3×3 matrices M_1-M_4 and their corresponding tree representations T_1-T_4 are given in Figure 3. M_1-M_3 obey MWP and hence T_1-T_3 are TOPs. T_4 is not a TOP because it has more than 3 leaves.

▶ **Lemma 14** (Numeric Bound). $|\mathcal{T}^r(n,k)| = 2^{O(n \lg n)}$.

Since $|\mathcal{R}^{\mu\mathrm{R}}| \leq n^n \cdot |\mathcal{M}^{\mu\mathrm{R}}|$, $|\mathcal{M}^{\mu\mathrm{R}}| \leq |\mathcal{M}^r| \cdot |\mathcal{M}^h|$, $|\mathcal{M}^r| \leq |\mathcal{T}^r(n,k)|$, and $|\mathcal{M}^h| \leq (H(n,k))^n$, by Lemmas 12 and 14, we have

▶ **Theorem 15** (Streett Upper Bound). *Streett complementation is* $2^{O(n \lg n + nk \lg k)}$ *for* $k = O(n)$ *and* $2^{O(n^2 \lg n)}$ *for* $k = \omega(n)$.

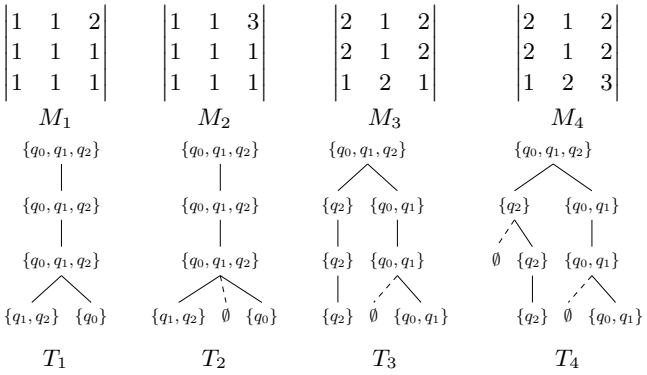


Figure 3 Four 3×3 matrices and their corresponding tree representations in Example 13.

Note that we put bounds in the form $2^{O(\cdot)}$ just for simplicity. Even for a small k (i.e. $k = O(n)$), our upper bound is substantially smaller than the current best one $(nk)^{O(nk)}$, established by respective constructions in [8, 22, 14, 19]. Easily seen from the proofs of Lemmas 12 and 14, our upper bound is in fact $n^{O(n)} \cdot k^{O(nk)}$ when $k = O(n)$. Also note that Lemma 14 is crucial in tightening parity complementation.

6 Parity Complementation

Parity automata is a special kind of Streett automata where a Streett condition $\langle G, B\rangle_I$ is augmented with the so-called *Rabin chain condition* $B(1) \subset G(1) \subset \cdots \subset B(k) \subset G(k)$. Now the short length of μR ranks is not enough to give us a better bound, because we already have $k \leq \lfloor (n+1)/2 \rfloor$. Nevertheless, the Rabin chain condition makes the GC condition $[B]_I$ degenerate to the CB condition $[B(1)]$, because being $B(1)$-free is equivalent to being $B(i)$-free for some $i \in I$. This coincides with the way Mini works. Intuitively, Mini synchronizes all components at a stage of μR ranking assignment. In the first stage of μR ranking assignment, in Step (4.1), Mini makes every vertex get the h-rank 1, though vertices may get different r-ranks. After disabling $G(1)$ vertices (by deleting all outgoing edges from them in Step (4.4)), we have a collection of components satisfying parity condition $\langle G, B\rangle_{[2..k]}$. Then in the second stage of μR ranking assignment, Mini gives every vertex the h-rank 2. Repeating this process, the h-projection of a final μR rank is just $\langle 1, \ldots, m\rangle$ for some $m \in [1..k]$, which is completely redundant, because the only useful information (having length m) is already encoded by the corresponding r-projection. As a consequence, h-matrices contribute nothing to the complexity. A customized construction for parity complementation is given in the appendix.

▶ **Theorem 16** (Parity Upper Bound). *Parity complementation is in* $2^{O(n \lg n)}$.

This bound matches the lower bound of Büchi complementation, and hence it is tight as Büchi automata are a subclass of parity automata. To the best of our knowledge, the previous best upper bound is $2^{O(nk \lg n)}$, which can be easily inferred from [14] by treating parity automata as Rabin automata.

7 Concluding Remarks

In this paper we improved Kupferman and Vardi's construction and obtained tight upper bounds for Streett and parity complementation (with respect to the $2^{\Theta(X)}$ asymptotic notation). Figure 1 in the appendix rounds up the complementation complexities for ω-automata of common types.

Our inquiry also leads to some unexpected outcomes, which we believe, would help understand the strength and weakness of different types of ω-automata in modeling and specifying system behaviors.

1. Parity complementation has the same asymptotical bound as Büchi complementation while parity automata have richer and more elegant acceptance conditions than Büchi automata.
2. Streett automata are exponentially more succinct than Büchi automata while Rabin automata are not. On the other hand, Streett complementation is much easier than Rabin complementation when k is large (i.e., $k = \omega(n)$). In the extreme case where $k = \Theta(2^n)$ and $N = \Theta(nk)$ (the automata size), Streett complementation is in $O(N^{\lg^2 N}) = O(2^{\lg^3 N})$ while Rabin complementation is still in $2^{\Omega(N)}$.

Further investigation on Streett and parity complementation is desired as exponential gaps can hide in the asymptotical notations of the form $2^{\Theta(X)}$. The situation is different from that of Büchi where the best lower and upper bounds have been shown polynomially close.

We think that ITS and TOP characterize intrinsic combinatorial properties on run graphs with universal Rabin conditions. Interesting questions remain for further investigation. What would be the counterparts for run graphs with existential Streett conditions? The discovery of such combinatorial properties might help us understand the complexity of Streett determinization, for which there exists a huge gap between the current lower bound $2^{\Omega(n^2 \lg n)}$ [3] and upper bound $2^{O(nk \lg nk)}$ [19] when $k = \omega(n)$. Also of theoretical interest is whether there exists a type of ω-automata whose determinization is considerably harder than complementation. In the case of Büchi, the two operations were both proved to be in $2^{\Theta(n \lg n)}$.

Acknowledgment

We would like to thank anonymous reviewers for many useful comments, and we are grateful to Laurel Tweed and Wanwu Wang for carefully proofreading the paper.

References

1 J.R. Büchi. On a decision method in restricted second order arithmetic. In Proc.*Internat. Congr. Logic, Method. and Philos. Sci.* 1960, pages 1-12, Stanford, 1962. Stanford University Press.

2 Y. Cai, T. Zhang, and H. Luo. An improved lower bound for the complementation of Rabin automata. In *Proc. 24th LICS*, pages 167-176, 2009.

3 Y. Cai and T. Zhang. A Tight lower bound for Streett complementation. Manuscript at `arXiv:1102.2963 [cs.LO]`.

4 Nachum Dershowitz and Shmuel Zaks. Enumerations of ordered tress. *Discrete Mathematics*, Vol. 31, No. 1 (1980) 9-28.

5 N. Francez and D. Kozen. Generalized fair termination. In *Proc. 11th POPL*, pages 46-53, 1984.

6 E. Friedgut and O. Kupferman and M.Y. Vardi. Büchi complementation made tighter. *International Journal of Foundations of Computer Science*, Vol. 17, No. 4 (2006) 851-867.

7 N. Francez. Fairness. *Texts and Monographs in Computer Science.* Springer-Verlag, 1986.

8 N. Klarlund. Progress measures for complementation of omega-automata with applications to temporal logic. In *Proc. 32th FOCS*, pages 358-367, 1991.

9 O. Kupferman. Avoiding Determinization. In *Proc. 21th LICS*, pages 243-254, 2006.

10 R.P. Kurshan. Computer aided verification of coordinating processes: an automata theoretic approach. Princeton University Press, 1994.

11 O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(3): 408-429, 2001.

12 O. Kupferman and M.Y. Vardi. From complementation to certification. In *10th TACAS*, LNCS 2988, pages 591-606, 2004.

13 O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th FOCS*, pages 531-540, 2005.

14 O. Kupferman and M.Y. Vardi. Complementation constructions for nondeterministic automata on infinite words. In *Proc. 11th TACAS*, pages 206-221, 2005.

15 C. Löding. Optimal bounds for transformations of omega-automata. In *Proc. 19th FSTTCS*, volume 1738 of *LNCS*, pages 97-109, 1999.

16 S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32(3):321-330, 1984.

17 M. Michel. Complementation is more difficult with automata on infinite words. *CNET*, Paris, 1988.

18 T. V. Narayana. A Partial Order and Its Applications to Probability Theory. *Sankhyā: The Indian Journal of Statistics (1933-1960)*, Vol. 21, (1959) 91-98.

19 N. Piterman. From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata. In *Proc. 21th LICS*, pages 255-264, 2006.

20 M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115-125, 1959.

21 S. Safra. On the complexity of ω-automata. In *Proc. 29th FOCS*, pages 319-327, 1988.

22 S. Safra. Exponential Determinization for ω-Automata with Strong-Fairness Acceptance Condition. In *Proc. 24th STOC*, pages 275-327, 1992.

23 S. Schewe. Büchi complementation made tight. In *Proc. 26th STACS*, pages 661-672, 2009.

24 A. P. Sistla, M.Y. Vardi, and P.Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217-327, 1987.

25 S. Safra and M.Y. Vardi. On ω-Automata and Temporal Logics. In *Proc. 29th STOC*, pages 127-137, 1989.

26 M.Y. Vardi. The Büchi complementation saga. In *Proc. 24th STACS*, pages 12-22, 2007.

27 M.Y. Vardi. and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st LICS*, pages 332-334, 1986.

28 Q. Yan. Lower bound for complementation of ω-automata via the full automata technique. In *Proc. 33th ICALP*, volume 4052 of *LNCS*, pages 589-600, 2006.

A Decidable Quantified Fragment of Set Theory Involving Ordered Pairs with Applications to Description Logics

Domenico Cantone, Cristiano Longo, and Marianna Nicolosi Asmundo

Dipartimento di Matematica e Informatica, Università di Catania
Viale Andrea Doria 6, 95125, Catania, Italy
{cantone|longo|nicolosi}@dmi.unict.it

Abstract

We present a decision procedure for a quantified fragment of set theory, called $\forall_{\mathbf{0}}^{\pi}$, involving ordered pairs and some operators to manipulate them. When our decision procedure is applied to $\forall_{\mathbf{0}}^{\pi}$-formulae whose quantifier prefixes have length bounded by a fixed constant, it runs in nondeterministic polynomial-time.

Related to the fragment $\forall_{\mathbf{0}}^{\pi}$, we also introduce a description logic, $\mathcal{DL}\langle\forall_{\mathbf{0}}^{\pi}\rangle$, which provides an unusually large set of constructs, such as, for instance, Boolean constructs among roles. The set-theoretic nature of the description logics semantics yields a straightforward reduction of the knowledge base consistency problem for $\mathcal{DL}\langle\forall_{\mathbf{0}}^{\pi}\rangle$ to the satisfiability problem for $\forall_{\mathbf{0}}^{\pi}$-formulae with quantifier prefixes of length at most 2, from which the NP-completeness of reasoning in $\mathcal{DL}\langle\forall_{\mathbf{0}}^{\pi}\rangle$ follows. Finally, we extend this reduction to cope with SWRL rules.

1998 ACM Subject Classification I.2.3 Deduction and Theorem Proving; F.4.1 Mathematical Logic; I.2.4 Knowledge Representation Formalisms and Methods

Keywords and phrases NP-complete decision procedures, set theory, description logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.129

1 Introduction

Computable Set Theory is a research field, started around thirty years ago, devoted to the study of the decision problem for fragments of set theory (see [4, 7] for a thorough account of the state-of-the-art until 2001). The most efficient decision procedures devised in this context have been implemented in the inferential core of the system ÆtnaNova/Referee, described in [8, 17, 19].

The first unquantified sublanguage of set theory that has been proved decidable is *Multi-Level Syllogistic* (in short MLS). MLS involves the set predicates $\in$, $\subseteq$, $=$, the Boolean set operators $\cup$, $\cap$, $\setminus$, and the connectives of propositional logic (cf. [10]). Subsequently, several extensions of MLS with various combinations of operators (such as singleton, powerset, unionset, etc.) and predicates (on finiteness, transitivity, etc.) have been proved to have a solvable satisfiability problem. Also, some extensions of MLS with various map[1] constructs have been shown to be decidable (cf. [9, 5]).

Concerning *quantified* fragments, of particular interest to us is the restricted quantified fragment of set theory $\forall_0$, which has been proved to have a decidable satisfiability problem

[1] According to [20], we use the term 'maps' to denote sets of ordered pairs.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 129–143

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

in [2]. We recall that $\forall_0$-formulae are propositional combinations of restricted quantified prenex formulae $(\forall y_1 \in z_1) \cdots (\forall y_n \in z_n)p$, where p is a Boolean combination of atoms of the forms $x \in y$, $x = y$, and no z_j is a y_i (i.e., nesting among quantified variables is not allowed). The same paper considered also the extension with another sort of variables representing *single-valued* maps, the map domain operator, and terms of the form $f(t)$ (representing the value of the map f on a function-free term t). However, neither one-to-many, nor many-to-one, nor many-to-many relationships can be represented in this language. We observe that the $\forall_0$-fragment is very close to the undecidability boundary, as shown in [18]. In fact, if nesting among quantified variables in prenex formulae of type $(\forall y_1 \in z_1) \cdots (\forall y_n \in z_n)p$ are allowed and a predicate stating that a set is an unordered pair is also admitted, then it turns out that the satisfiability for the resulting collection of formulae is undecidable.

In this paper we present a decision procedure for the novel fragment of set theory $\forall_{\mathbf{0}}^{\pi}$, which extends the fragment $\forall_0$ with ordered pairs and various constructs related to them, thus further thinning the gap between the decidable and the undecidable. A considerable amount of set-theoretic constructs can be expressed by $\forall_{\mathbf{0}}^{\pi}$-formulae, in particular constructs on *multi-valued* maps like map inverse, Boolean operator among maps, map transitivity, and so on. Furthermore, when restricted to formulae with quantifier nesting bounded by a constant, our decision procedure runs in nondeterministic polynomial-time. This fragment has also interesting applications in the field of *knowledge representation*.

Applications of Computable Set Theory to knowledge representation have been recently proposed in [6], where the correspondence between (decidable) fragments of set theory and *Description Logics* (a well established framework for knowledge representation systems; see [1] for a quite complete overview) is exploited by introducing the very expressive description logic $\mathcal{DL}\langle \mathsf{MLSS}^{\times}_{2,m} \rangle$.

Description logics are a family of logic based formalisms widely used in knowledge representation. In particular, several results and decision procedures devised in this context have been profitably employed in the area of the Semantic Web (cf. [11]). The key problem in description logic is to determine whether a knowledge base $\mathcal{K}$ is *consistent* (knowledge base consistency is formally described in Section 4), and many other reasoning tasks can be reduced to it. Unfortunately, this problem is ExpTime-hard (cf. [1, Theorem 3.27, page 132]) also for $\mathcal{AL}$, a basic description logic with a very limited expressive power. However, [6] shows how a better computational complexity can be achieved by imposing some limitations on the usage of existential quantification and number restrictions (definitions of these two constructs are reported in Table 1).

The quantified nature of the language $\forall_{\mathbf{0}}^{\pi}$ and the pair-related constructs it provides allow a straightforward mapping of numerous description logic constructs to $\forall_{\mathbf{0}}^{\pi}$-formulae. The resulting description logic, called $\mathcal{DL}\langle \forall_{\mathbf{0}}^{\pi} \rangle$, extends those presented in [6] with several constructs like, for instance, role transitivity, self restrictions, and role identity. It also allows *finite* existential restrictions of the form $\exists R.\{a_1, \ldots, a_n\}$ to be used without limitations. Furthermore, it turns out that the consistency problem for $\mathcal{DL}\langle \forall_{\mathbf{0}}^{\pi} \rangle$-knowledge bases is NP-complete. This is a quite significant result since in most of the cases in which Boolean operators among roles are present the consistency problem turns out to be NExpTime-hard (cf. [14]).

Finally, we observe that SWRL rules (cf. [12]) can be easily embedded in $\mathcal{DL}\langle \forall_{\mathbf{0}}^{\pi} \rangle$ without disrupting decidability. SWRL rules are a simple form of Horn-style rules, which were proposed with the aim of increasing the expressive power of description logics. Here we consider only a restricted set of SWRL rules, namely those which do not contain *data literals*. Extending description logics with SWRL rules in general leads to undecidability. In [16] this

issue has been overcome by restricting the applicability of rules to a finite set of named individuals. Another approach, studied in [13], consists in restricting to rules which can be *internalized*, i.e. rules which can be converted into knowledge base statements.

The paper is organized as follows. Section 2 presents the precise syntax and semantics of the language $\forall^{\pi}_{\mathbf{0}}$. A decision procedure for $\forall^{\pi}_{\mathbf{0}}$ is then developed in Section 3. In Section 4 the correspondences of $\forall^{\pi}_{\mathbf{0}}$ with description logics are exploited by introducing the novel description logic $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$, whose extension with SWRL rules is studied in Section 5. Finally, concluding remarks and some hints to future work are given in Section 6.

2 The language $\forall^{\pi}_{\mathbf{0}}$

The language $\forall^{\pi}_{\mathbf{0}}$ is a quantified fragment of set theory which contains a denumerable infinity of *variables*, $Vars = \{x, y, z, \ldots\}$, the binary *pairing* operator $[\cdot,\cdot]$, the monadic function $\bar{\pi}(\cdot)$, which represents the non-pair members of a set, the relators $\in, =$, the Boolean connectives of propositional logic $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, parentheses, and the universal quantifier $\forall$.

A *quantifier-free* $\forall^{\pi}_{\mathbf{0}}$*-formula* is any propositional combination of *atomic* $\forall^{\pi}_{\mathbf{0}}$*-formulae*. These are expressions of the following types:

$$x \in \bar{\pi}(z), \quad [x, y] \in z, \quad x = y, \tag{1}$$

with $x, y, z \in Vars$. Intuitively, terms of the form $[x, y]$ represent ordered pairs of sets.

A *simple prenex* $\forall^{\pi}_{\mathbf{0}}$*-formula* is a formula $Q_1 \cdots Q_n \varphi$, with $n \geq 0$, where φ is a quantifier-free $\forall^{\pi}_{\mathbf{0}}$-formula, each Q_i is a restricted universal quantifier of form $(\forall x \in \bar{\pi}(y))$ or of the form $(\forall [x, x'] \in y)$ (we will refer to x and x' as *quantified variables* and to y as *domain variable*), and no variable can occur both as a quantified and a domain variable, i.e., roughly speaking, no x can be a y.

Finally, a $\forall^{\pi}_{\mathbf{0}}$*-formula* is any finite conjunction of simple prenex $\forall^{\pi}_{\mathbf{0}}$-formulae.

Semantics of the $\forall^{\pi}_{\mathbf{0}}$-language is based upon the von Neumann standard cumulative hierarchy $\mathcal{V}$ of sets, which is defined as follows:

$$\begin{array}{rcll} \mathcal{V}_0 & = & \emptyset & \\ \mathcal{V}_{\gamma+1} & = & \mathcal{P}(\mathcal{V}_\gamma)\,, & \text{for each ordinal } \gamma \\ \mathcal{V}_\lambda & = & \bigcup_{\mu<\lambda} \mathcal{V}_\mu\,, & \text{for each limit ordinal } \lambda \\ \mathcal{V} & = & \bigcup_{\gamma \in On} \mathcal{V}_\gamma\,, & \end{array}$$

where $\mathcal{P}(\cdot)$ is the powerset operator and On denotes the class of all ordinals.

A $\forall^{\pi}_{\mathbf{0}}$*-interpretation* is a pair $\mathbf{I} = (M_{\mathbf{I}}, \pi_{\mathbf{I}})$, where $M_{\mathbf{I}}$ is a total function which maps each variable into a set of $\mathcal{V}$, and $\pi_{\mathbf{I}}$ is a *pairing function* over sets. We recall that a *pairing function* π is a binary operation over sets such that $\pi(u, v) = \pi(u', v') \iff u = u' \wedge v = v'$ and the class $u \times_\pi v =_{\text{Def}} \{\pi(s, t) : s \in u \wedge t \in v\}$ is a set of $\mathcal{V}$, for all $u, v, u', v' \in \mathcal{V}$.

Let W be a finite subset of $Vars$, we say that $\mathbf{I}' = (M_{\mathbf{I}'}, \pi_{\mathbf{I}})$ is a W*-variant* of $\mathbf{I}$ if $M_{\mathbf{I}'} y = M_{\mathbf{I}} y$, for $y \in Vars \backslash W$. To any term of the form x, $[x, y]$, and $\bar{\pi}(x)$, a $\forall^{\pi}_{\mathbf{0}}$-interpretation $\mathbf{I}$ associates a set in $\mathcal{V}$ as follows:

$$\begin{array}{rcl} \mathbf{I}x & =_{\text{Def}} & M_{\mathbf{I}} x \\ \mathbf{I}[x, y] & =_{\text{Def}} & \pi_{\mathbf{I}}(\mathbf{I}x, \mathbf{I}y) \\ \mathbf{I}\bar{\pi}(x) & =_{\text{Def}} & \mathbf{I}x \setminus \{\pi_{\mathbf{I}}(u, v) : u, v \in \mathcal{V}\}, \end{array}$$

for all $x, y \in Vars$.

A $\forall^{\pi}_{\mathbf{0}}$-interpretation evaluates atomic $\forall^{\pi}_{\mathbf{0}}$-formulae to the truth values **t** (true) and **f** (false) in the usual way, by interpreting '$\in$' and '$=$' as the membership and the equality relations

between sets, respectively. Evaluation of quantifier-free $\forall_{\mathbf{0}}^{\pi}$-formulae is carried out according to the standard rules of propositional logic, and simple prenex $\forall_{\mathbf{0}}^{\pi}$-formulae are evaluated as follows:

- $\mathbf{I}(\forall x \in \bar{\pi}(y))\varphi = \mathbf{t}$ iff $\mathbf{I}'\varphi = \mathbf{t}$ for every $\{x\}$-variant $\mathbf{I}'$ of $\mathbf{I}$ such that $\mathbf{I}'x \in \mathbf{I}'\bar{\pi}(y)$,
- $\mathbf{I}(\forall [x, y] \in z)\varphi = \mathbf{t}$ iff $\mathbf{I}'\varphi = \mathbf{t}$ for every $\{x, y\}$-variant $\mathbf{I}'$ of $\mathbf{I}$ such that $\mathbf{I}'[x, y] \in \mathbf{I}'z$.

A $\forall_{\mathbf{0}}^{\pi}$-interpretation $\mathbf{I}$ which evaluates a $\forall_{\mathbf{0}}^{\pi}$-formula φ to true is said to be a *model* for φ (and we write $\mathbf{I} \models \varphi$). A $\forall_{\mathbf{0}}^{\pi}$-formula is said to be *satisfiable* if it admits a model. Thus the satisfiability problem (in short, s.p.) for $\forall_{\mathbf{0}}^{\pi}$-formulae consists in determining whether a $\forall_{\mathbf{0}}^{\pi}$-formula is satisfiable or not. Observe that in the context of satisfiability, all free variables in a $\forall_{\mathbf{0}}^{\pi}$-formula may be regarded as existentially quantified.

In the following section we present a decision procedure for the s.p. for $\forall_{\mathbf{0}}^{\pi}$-formulae.

3 A decision procedure for $\forall_{\mathbf{0}}^{\pi}$

In this section we solve the s.p. for $\forall_{\mathbf{0}}^{\pi}$-formulae. In particular, we will prove that a $\forall_{\mathbf{0}}^{\pi}$-formula is satisfiable if and only if there exists a finite collection of atomic $\forall_{\mathbf{0}}^{\pi}$-formulae which *represents* a model for the formula. We begin by introducing the notions of skeletal representations and of their realizations: these are, respectively, collections of atomic $\forall_{\mathbf{0}}^{\pi}$-formulae with an acyclic membership relation among their variables, and suitably defined $\forall_{\mathbf{0}}^{\pi}$-interpretations. In particular, we will focus on skeletal representations "completed" w.r.t. the predicate "=" over a set of variables V, which we call V-extensional. It turns out, as will be shown in Lemma 2, that each V-extensional skeletal representation is modeled correctly by any realization associated with it. Finally, we prove the main result of this section, namely that a $\forall_{\mathbf{0}}^{\pi}$-formula φ with free variables V is satisfiable if and only if it is satisfied by the realization associated with a suitable V-extensional skeletal representation whose size is bounded by the cardinality of V (cf. Theorem 3). The latter result entails immediately the decidability of the fragment $\forall_{\mathbf{0}}^{\pi}$ of our interest.

Given a $\forall_{\mathbf{0}}^{\pi}$-formula φ, we denote with φ_y^x the formula obtained by replacing each free occurrence of x in φ with y and with $\mathit{Vars}(\varphi)$ the collection of the free variables of φ. Likewise, given a finite collection $\mathcal{S}$ of atomic $\forall_{\mathbf{0}}^{\pi}$-formulae, we denote with $\mathit{Vars}(\mathcal{S})$ the collection of the variables occurring in the formulae of $\mathcal{S}$. In addition, we indicate with $\in_{\mathcal{S}}^{+}$ (the *membership closure* of $\mathcal{S}$) the minimal transitive relation on $\mathit{Vars}(\mathcal{S})$ such that the following conditions hold:

- if "$x \in \bar{\pi}(z)$" $\in \mathcal{S}$, then $x \in_{\mathcal{S}}^{+} z$;
- if "$[x, y] \in z$" $\in \mathcal{S}$, then $x \in_{\mathcal{S}}^{+} z \wedge y \in_{\mathcal{S}}^{+} z$.

A collection $\mathcal{S}$ of atomic $\forall_{\mathbf{0}}^{\pi}$-formulae is a *skeletal representation* if $x \notin_{\mathcal{S}}^{+} x$, for all $x \in \mathit{Vars}(\mathcal{S})$.

Let $\mathcal{S}$ be a skeletal representation. We define the *height* of a variable $x \in \mathit{Vars}(\mathcal{S})$ with respect to $\mathcal{S}$ (which we write $\mathsf{height}_{\mathcal{S}}(x)$) as the length n of the longest $\in_{\mathcal{S}}^{+}$-chain of the form $x_1 \in_{\mathcal{S}}^{+} \ldots \in_{\mathcal{S}}^{+} x_n \in_{\mathcal{S}}^{+} x$ ending at x, with $x_1, \ldots, x_n \in \mathit{Vars}(\mathcal{S})$. Thus, $\mathsf{height}_{\mathcal{S}}(x) = 0$ if $y \notin_{\mathcal{S}}^{+} x$, for all $y \in \mathit{Vars}(\mathcal{S})$.

A skeletal representation $\mathcal{S}$ is said to be *V-extensional*, for a given set of variables V, if the following conditions hold:

- if "$x = y$" $\in \mathcal{S}$, then $x, y \in V$ and α_y^x and α_x^y belong to $\mathcal{S}$, for each atomic formula α in $\mathcal{S}$;
- if "$x = y$" $\notin \mathcal{S}$, for some $x, y \in V$, then the variables x and y must be explicitly *distinguished* in $\mathcal{S}$ either by some variable z, in the sense that "$z \in \bar{\pi}(x)$" $\in \mathcal{S}$ iff "$z \in \bar{\pi}(y)$" $\notin \mathcal{S}$, or by some pair $[z, z']$, in the sense that "$[z, z'] \in x$" $\in \mathcal{S}$ iff "$[z, z'] \in y$" $\notin \mathcal{S}$.

Skeletal representations allow one to define special $\forall_{\mathbf{0}}^{\pi}$-interpretations, called *realizations*, which were first introduced in [3], though with a slightly different meaning. To this purpose we introduce the following family $\{\pi_n\}_{n\in\mathbb{N}}$ of pairing functions, recursively defined by

$$\begin{array}{rcl} \pi_0(u,v) & =_{\text{Def}} & \{u,\{u,v\}\} \\ \pi_{n+1}(u,v) & =_{\text{Def}} & \{\pi_n(u,v)\}\,, \end{array}$$

for every $u, v \in \mathcal{V}$.

▶ **Definition 1** (Realization)**.** Let $\mathcal{S}$ be a skeletal representation, let V and T be two finite, nonempty, and disjoint sets of variables such that $\mathit{Vars}(\mathcal{S}) \subseteq V \cup T$, and let σ be a bijection from T onto $\{1, 2, \ldots, |T|\}$. We extend the function $\mathsf{height}_{\mathcal{S}}(\cdot)$ also to variables $x \in (V \cup T) \setminus \mathit{Vars}(\mathcal{S})$ by putting for such variables $\mathsf{height}_{\mathcal{S}}(x) =_{\text{Def}} 0$.

Then the *realization* of $\mathcal{S}$ relative to (V, T) is the $\forall_{\mathbf{0}}^{\pi}$-interpretation $\mathbf{R} = (M_{\mathbf{R}}, \pi_{\mathbf{R}})$ such that $\pi_{\mathbf{R}} =_{\text{Def}} \pi_{|V|+|T|}$ and, recursively on $\mathsf{height}_{\mathcal{S}}(x)$ for $x \in V \cup T$,

$$M_{\mathbf{R}}x =_{\text{Def}} \{\mathbf{R}y : \text{“}y \in \bar{\pi}(x)\text{”} \in \mathcal{S}\} \cup \{\mathbf{R}[y,z] : \text{“}[y,z] \in x\text{”} \in \mathcal{S}\} \cup s(x)\,,$$

where

$$s(x) =_{\text{Def}} \begin{cases} \{\{k+1, k, \sigma(x)\}\} & \text{if } x \in T \\ \emptyset & \text{otherwise,} \end{cases}$$

with $k = |V| \cdot (|V| + |T| + 3)$.[2] ◀

Realizations have useful properties, stated by the following lemma.

▶ **Lemma 2.** *Let $\mathcal{S}$, V, T, σ, and k be as in Definition 1 and let $\mathbf{R}$ be the realization of $\mathcal{S}$ relative to (V, T). If $\mathcal{S}$ is V-extensional, then for every $x, y, z \in V \cup T$ the following conditions hold:*

(R1) *$\mathbf{R}x \neq \pi_{\mathbf{R}}(u, v)$ for $u, v \in \mathcal{V}$;*
(R2) *$\mathbf{R}x \neq \{k+1, k, i\}$ for $1 \leq i \leq |T|$;*
(R3) *$\mathbf{R}x = \mathbf{R}y$ iff either “$x = y$” $\in \mathcal{S}$ or x and y coincide;*
(R4) *$\mathbf{R}x \in \mathbf{R}\bar{\pi}(y)$ iff “$x \in \bar{\pi}(y)$” $\in \mathcal{S}$;*
(R5) *$\mathbf{R}[x, y] \in \mathbf{R}z$ iff “$[x, y] \in z$” $\in \mathcal{S}$.*

Proof. To prove **(R1)**, we establish the more general property

$$\text{if } \mathsf{height}_{\mathcal{S}}(x) \leq n \leq |V| + |T|, \quad \text{then } \mathbf{R}x \neq \pi_n(u, v), \text{ for } x \in V \cup T \text{ and } u, v \in \mathcal{V}. \tag{2}$$

Let $n \leq |V| + |T|$ and let us assume by way of contradiction that $\mathbf{R}x = \pi_n(u, v)$ for some $u, v \in \mathcal{V}$ and some $x \in V \cup T$ of minimal height such that $0 \leq \mathsf{height}_{\mathcal{S}}(x) \leq n$.

We can rule out at once the case in which $n = 0$, as in this case $\mathsf{height}_{\mathcal{S}}(x) = 0$, so that $|\mathbf{R}x| \leq 1$, and therefore $\mathbf{R}x \neq \pi_0(u, v)$, since $|\pi_0(u, v)| = 2$.

Thus, we can assume that $n > 0$. Let us consider first the case in which $\mathsf{height}_{\mathcal{S}}(x) = 0$. If $x \in V$ then, by the very definition of realization, we have $\mathbf{R}x = \emptyset \neq \pi_n(u, v)$. On the other hand, if $x \in T$, then $\mathbf{R}x = \{\{k+1, k, \sigma(x)\}\}$ and since $|\{k+1, k, \sigma(x)\}| > |\pi_{n-1}(u, v)|$ and $\pi_n(u, v) = \{\pi_{n-1}(u, v)\}$, it follows that $\mathbf{R}x \neq \pi_n(u, v)$. In both cases we found a contradiction, so that we must have $\mathsf{height}_{\mathcal{S}}(x) > 0$.

On the other hand, if $\mathsf{height}_{\mathcal{S}}(x) > 0$, our absurd hypothesis $\mathbf{R}x = \pi_n(u, v) = \{\pi_{n-1}(u, v)\}$ and the definition of realization imply that either

[2] We are assuming that integers are represented *à la* von Neumann, namely $0 =_{\text{Def}} \emptyset$ and, recursively, $n + 1 =_{\text{Def}} n \cup \{n\}$.

(i) $\pi_{n-1}(u,v) = \{k+1, k, \sigma(x)\}$, but provided that $x \in T$, or
(ii) $\pi_{n-1}(u,v) = \mathbf{R}y$, for some y such that "$y \in \bar{\pi}(x)$" $\in \mathcal{S}$, or
(iii) $\pi_{n-1}(u,v) = \mathbf{R}[y,z] = \pi_{|V|+|T|}(\mathbf{R}y, \mathbf{R}z)$, for some y, z such that "$[y,z] \in x$" $\in \mathcal{S}$.

We can exclude at once case **(i)**, since $|\pi_{n-1}(u,v)| \leq 2 < |\{k+1, k, \sigma(x)\}|$. Case **(ii)** can be excluded as well, since it would contradict the minimality of $\mathsf{height}_{\mathcal{S}}(x)$, as $\mathsf{height}_{\mathcal{S}}(y) < \mathsf{height}_{\mathcal{S}}(x)$. In case **(iii)**, from elementary properties of our pairing functions π_i it would follow that $|V| + |T| = n - 1$, contradicting our initial assumption that $n \leq |V| + |T|$. Thus (2) holds.

In view of (2), to establish **(R1)** it is now enough to observe that $\mathsf{height}_{\mathcal{S}}(x) < |V| + |T|$.

Next, since $\mathsf{rank}(\{k+1, k, i\}) = k+2$, for $1 \leq i \leq |T|$ (as $k > |T|$),[3] to establish **(R2)** it will be enough to show that $\mathsf{rank}(\mathbf{R}x) \neq k+2$, for $x \in V \cup T$. Thus, let $x \in V \cup T$. If $y \in^+_{\mathcal{S}} x$, for some $y \in T$, then $\mathsf{rank}(\mathbf{R}x) \geq \mathsf{rank}(\mathbf{R}y) \geq k+3$. The same conclusion can be reached also in the case in which $x \in T$. On the other hand, if $y \notin^+_{\mathcal{S}} x$, for any $y \in T$, and $x \in V$, it can easily be proved by induction on $\mathsf{height}_{\mathcal{S}}(x)$ that $\mathsf{rank}(\mathbf{R}x) \leq (|V| + |T| + 3) \cdot \mathsf{height}_{\mathcal{S}}(x) \leq |V| \cdot (|V| + |T| + 3) = k$. Hence, in any case $\mathsf{rank}(\mathbf{R}x) \neq k+2$ holds, proving **(R2)**.

Concerning **(R3)**, we observe preliminarily that if "$x = y$" $\in \mathcal{S}$, then $\mathbf{R}x = \mathbf{R}y$ is a direct consequence of the V-extensionality of $\mathcal{S}$. Thus, to complete the proof of **(R3)** it is enough to show that if $\mathbf{R}x = \mathbf{R}y$, for distinct variables $x, y \in V \cup T$, then "$x = y$" $\in \mathcal{S}$. So, assume that "$x = y$" $\notin \mathcal{S}$, for two distinct variables $x, y \in V \cup T$ and consider first the case in which either x or y, say y, is a variable in T. From the definition of realization it follows that $\{k+1, k, \sigma(y)\} \in \mathbf{R}y$, while from **(R2)** and the fact that $\{k+1, k, \sigma(y)\}$ is not a pair with respect to $\pi_{|V|+|T|}$, it follows that $\{k+1, k, \sigma(y)\} \notin \mathbf{R}x$, unless $x \in T$ and $\{k+1, k, \sigma(y)\} = \{k+1, k, \sigma(x)\}$. But in such a case, we would have $\sigma(x) = \sigma(y)$ and therefore x and y must coincide, contradicting our initial assumption that x and y are distinct variables. Therefore we have $\mathbf{R}x \neq \mathbf{R}y$.

Next, let us assume that $x, y \in V$. We will induction on $\max(\mathsf{height}_{\mathcal{S}}(x), \mathsf{height}_{\mathcal{S}}(y))$. From the V-extensionality of $\mathcal{S}$ it follows that x, y are distinguished in $\mathcal{S}$ by a variable z or by a pair $[z', z'']$. Let us first assume that x, y are distinguished in $\mathcal{S}$ by a variable z. If "$z \in \bar{\pi}(x)$" $\in \mathcal{S}$ and "$z \in \bar{\pi}(y)$" $\notin \mathcal{S}$, then for all w such that "$w \in \bar{\pi}(y)$" $\in \mathcal{S}$ we have $\mathbf{R}z \neq \mathbf{R}w$ by the inductive hypothesis, since $\mathsf{height}_{\mathcal{S}}(z) < \mathsf{height}_{\mathcal{S}}(x)$ and $\mathsf{height}_{\mathcal{S}}(w) < \mathsf{height}_{\mathcal{S}}(y)$. Furthermore, from **(R1)** it follows also that $\mathbf{R}z \neq \mathbf{R}[w, w']$, for all w, w' such that "$[w, w'] \in y$" $\in \mathcal{S}$. Thus $\mathbf{R}z \in \mathbf{R}x \setminus \mathbf{R}y$. If "$z \in \bar{\pi}(y)$" $\in \mathcal{S}$ and "$z \in \bar{\pi}(x)$" $\notin \mathcal{S}$ we can prove that $\mathbf{R}z \in \mathbf{R}y \setminus \mathbf{R}x$ in an analogous way. In both case we have $\mathbf{R}x \neq \mathbf{R}y$. On the other hand, if x, y are distinguished by a pair $[z', z'']$, we can argue as follows. Assume first that "$[z', z''] \in x$" $\in \mathcal{S}$ and "$[z', z''] \in y$" $\notin \mathcal{S}$. Plainly, $\mathbf{R}[z', z''] \in \mathbf{R}x$. If $\mathbf{R}[z', z''] \in \mathbf{R}y$, then by **(R1)** $\mathbf{R}[z', z''] = \mathbf{R}[w', w'']$, for a pair $[w', w'']$ such that "$[w', w''] \in y$" $\in \mathcal{S}$. Since $\pi_{|V|+|T|}$ is a pairing function, we have $\mathbf{R}z' = \mathbf{R}w'$ and $\mathbf{R}z'' = \mathbf{R}w''$. Considering that $\mathsf{height}_{\mathcal{S}}(z'), \mathsf{height}_{\mathcal{S}}(z'') < \mathsf{height}_{\mathcal{S}}(x)$ and that $\mathsf{height}_{\mathcal{S}}(w'), \mathsf{height}_{\mathcal{S}}(w'') < \mathsf{height}_{\mathcal{S}}(y)$, the inductive hypothesis yields that

- z' and w' coincide or "$z' = w'$" is in $\mathcal{S}$, and
- z'' and w'' coincide or "$z'' = w''$" is in $\mathcal{S}$.

But then, by the V-extensionality of $\mathcal{S}$, "$[z', z''] \in y$" would be in $\mathcal{S}$, a contradiction. Hence, $\mathbf{R}[z', z''] \in \mathbf{R}x \setminus \mathbf{R}y$. Analogously, if "$[z', z''] \in x$" $\notin \mathcal{S}$ and "$[z', z''] \in y$" $\in \mathcal{S}$, we have $\mathbf{R}[z', z''] \in \mathbf{R}y \setminus \mathbf{R}x$. Therefore, in both cases we have $\mathbf{R}x \neq \mathbf{R}y$, proving **(R3)**.

[3] We recall that the *rank* of a set $u \in \mathcal{V}$ denotes the least ordinal γ such that $u \subseteq \mathcal{V}_\gamma$ (i.e., $u \in \mathcal{V}_{\gamma+1}$).

The cases **(R4)** and **(R5)** are easy consequences of **(R1)**, **(R2)**, and **(R3)**. Details are left to the reader. This completes the proof of the lemma. ◀

Realizations act as *minimal models* for skeletal representations, in the sense that if V, T are two disjoint sets of variables, $\mathcal{S}$ is a V-extensional skeletal representation such that $Vars(\mathcal{S}) \subseteq V \cup T$, and $\mathbf{R}$ is the realization of $\mathcal{S}$ relative to (V, T) (and to a bijection σ), then $\mathbf{R} \models \alpha$ if and only if $\alpha \in \mathcal{S}$.

In the next theorem we show how skeletal representations can be used to witness the satisfiability of $\forall_{\mathbf{0}}^{\pi}$-formulae.

▶ **Theorem 3.** *Let φ be a $\forall_{\mathbf{0}}^{\pi}$-formula, and let $V = Vars(\varphi)$. Then φ is satisfiable iff there exists a V-extensional skeletal representation $\mathcal{S}$ such that:*

(i) $Vars(\mathcal{S}) \subseteq V \cup T$, for some T such that $1 \leq |T| < 2|V|$;
(ii) $\mathbf{R} \models \varphi$, where $\mathbf{R}$ is the realization of $\mathcal{S}$ relative to (V, T).

Proof. To prove the theorem, it is enough to exhibit a skeletal representation $\mathcal{S}$ that satisfies conditions (i) and (ii) above, given a model $\mathbf{I}$ for φ.

Thus, let $\mathbf{I}$ be a model for φ and let $\Sigma = \{\mathbf{I}x : x \in V\}$. As shown in [3], there exists a collection Σ_0 of size strictly less than $|\Sigma|$ which witnesses all the inequalities among the members of Σ, in the sense that $s \cap \Sigma_0 \neq s' \cap \Sigma_0$ for any two distinct $s, s' \in \Sigma$. Let us *split* the pairs present in Σ_0 (relative to the pairing function $\pi_{\mathbf{I}}$ of $\mathbf{I}$) forming the collection

$$\Sigma_1 =_{\text{Def}} \{s : s \in \Sigma_0 \wedge (\forall u, v \in \mathcal{V})(s \neq \pi_{\mathbf{I}}(u, v))\} \cup \bigcup \{\{u, v\} : \pi_{\mathbf{I}}(u, v) \in \Sigma_0\}.$$

Then we put

$$\Sigma_2 =_{\text{Def}} \begin{cases} \Sigma_1 \setminus \Sigma & \text{if } \Sigma_1 \setminus \Sigma \neq \emptyset \\ \{\emptyset\} & \text{otherwise} \end{cases}$$

and let T be any collection of variables in *Vars*, not already occurring in φ, such that $|T| = |\Sigma_2|$ (so that $|T| \geq 1$). Notice that $|T| \leq 2|\Sigma_0| + 1 < 2|V|$.

Finally, we define our skeletal representation as the collection $\mathcal{S}$ of atomic $\forall_{\mathbf{0}}^{\pi}$-formulae such that:

$$\begin{aligned} \text{“}x \in \bar{\pi}(y)\text{”} \in \mathcal{S} &\iff \mathbf{I}x \in \mathbf{I}\bar{\pi}(y) \\ \text{“}[x, y] \in z\text{”} \in \mathcal{S} &\iff \mathbf{I}[x, y] \in \mathbf{I}z \\ \text{“}x = y\text{”} \in \mathcal{S} &\iff \mathbf{I}x = \mathbf{I}y \text{ and } x, y \in V \end{aligned}$$

for all $x, y, z \in V \cup T$.

As can be easily verified, the above construction process yields a V-extensional skeletal representation $\mathcal{S}$ satisfying condition (i) of the theorem.

We prove next that also condition (ii) is satisfied, i.e. $\mathbf{R} \models \varphi$ holds, where $\mathbf{R}$ is the realization of $\mathcal{S}$ relative to (V, T). This amounts to showing that $\mathbf{R}$ models correctly all conjuncts of φ. These are simple prenex $\forall_{\mathbf{0}}^{\pi}$-formulae whose free variables belong to $V \cup T$ and whose domain variables belong to V, which are correctly modeled by $\mathbf{I}$. It will therefore be enough to prove the following general property stating that

$$\mathbf{I} \models \psi \Longrightarrow \mathbf{R} \models \psi, \tag{3}$$

for every simple prenex $\forall_{\mathbf{0}}^{\pi}$-formula ψ such that $Vars(\psi) \subseteq V \cup T$ and whose domain variables, if any, belong to V.

We prove (3) by induction on the length of the quantifier prefix of ψ.

When ψ is quantifier-free, (3) follows from propositional logic, by observing that the definition of $\mathcal{S}$ together with conditions **(R3)**, **(R4)**, and **(R5)** of Lemma 2 yield that $\mathbf{I}\alpha = \mathbf{R}\alpha$, for each atomic $\forall^{\pi}_{\mathbf{0}}$-formula α such that $Vars(\alpha) \subseteq V \cup T$.

For the inductive step, let ψ have either the form $(\forall x \in \bar{\pi}(y))\chi$ or the form $(\forall [x, y] \in z)\chi$, with χ a simple prenex $\forall^{\pi}_{\mathbf{0}}$-formula having one less quantifier than ψ. For the sake of simplicity, we consider here only the case in which ψ has the form $(\forall x \in \bar{\pi}(y))\chi$, as the other case can be dealt with much in the same manner. We remark that, by hypothesis, the domain variable y in $(\forall x \in \bar{\pi}(y))\chi$ belongs to V.

Let us assume that $\mathbf{I} \models \psi$. To complete the inductive proof of (3) we need to show that $\mathbf{R} \models \psi$. From $\mathbf{I} \models \psi$ it follows that $\mathbf{I} \models (w \in \bar{\pi}(y)) \rightarrow \chi^x_w$, for every variable w, and in particular for every variable $w \in W$, where $W =_{\text{Def}} \{w \in V \cup T : \text{“}w \in \bar{\pi}(y)\text{”} \in \mathcal{S}\}$. Let $w \in W$. We clearly have $\mathbf{I} \models w \in \bar{\pi}(y)$, and therefore $\mathbf{I} \models \chi^x_w$. Plainly, $Vars(\chi^x_w) \subseteq V \cup T$. In addition, all domain variables in χ^x_w belong to V, since this is the case for all domain variables in χ and w can not appear in χ^x_w as a domain variable, since x is a quantified variable of ψ and as such can not appear also as a domain variable in ψ, and therefore in χ. Hence, by inductive hypothesis, we have $\mathbf{R} \models \chi^x_w$ and, *a fortiori*, $\mathbf{R} \models (w \in \bar{\pi}(y)) \rightarrow \chi^x_w$.

Notice that the latter relation holds also for $w \in (V \cup T) \setminus W$, since in this case $\mathbf{I} \not\models (w \in \bar{\pi}(y))$ and therefore, as observed above, $\mathbf{R} \not\models (w \in \bar{\pi}(y))$. Thus we have

$$\mathbf{R} \models (w \in \bar{\pi}(y)) \rightarrow \chi^x_w, \tag{4}$$

for every $w \in V \cup T$. We show that (4) implies $\mathbf{R} \models (\forall x \in \bar{\pi}(y))\chi$, which is what we want to prove.

Indeed, if by contradiction $\mathbf{R} \not\models (\forall x \in \bar{\pi}(y))\chi$, then $\mathbf{R}' \not\models (x \in \bar{\pi}(y)) \rightarrow \chi$, for some $\{x\}$-variant $\mathbf{R}'$ of $\mathbf{R}$, so that $\mathbf{R}' \models (x \in \bar{\pi}(y))$ and $\mathbf{R}' \not\models \chi$. But then

$$\mathbf{R}'x \in \mathbf{R}'\bar{\pi}(y) = \mathbf{R}\bar{\pi}(y) \subseteq \{\mathbf{R}z : \text{“}z \in \bar{\pi}(y)\text{”} \in \mathcal{S}\}.$$

Therefore $\mathbf{R}'x = \mathbf{R}z_0$, for some variable z_0 (in $V \cup T$) such that the literal “$z_0 \in \bar{\pi}(y)$” belongs to $\mathcal{S}$. Thus we have $\mathbf{R} \models z_0 \in \bar{\pi}(y)$ and $\mathbf{R} \not\models (z_0 \in \bar{\pi}(y)) \rightarrow \chi^x_{z_0}$, contradicting (4). Hence, $\mathbf{R} \models (\forall x \in \bar{\pi}(y))\chi$ holds, completing the inductive proof of (3) and, in turn, the proof of condition (ii) of the theorem. ◀

Theorem 3 yields a decision test for the s.p. for $\forall^{\pi}_{\mathbf{0}}$-formulae, as the number of possible V-extensional skeletal representations satisfying condition (i) of the theorem is finite, for any given $\forall^{\pi}_{\mathbf{0}}$-formula, and condition (ii) is effectively verifiable. In the following section, we analyze the s.p. for $\forall^{\pi}_{\mathbf{0}}$-formulae from a complexity point of view.

3.1 Complexity issues

The s.p. for propositional logic can be easily reduced to the one for $\forall^{\pi}_{\mathbf{0}}$-formulae as follows. Given a propositional formula Q, we construct in linear time a quantifier-free $\forall^{\pi}_{\mathbf{0}}$-formula φ_Q, by replacing each propositional variable p in Q with a corresponding atomic $\forall^{\pi}_{\mathbf{0}}$-formula $x_p \in \bar{\pi}(U)$, where U is a set variable distinct from all set variables x_p so introduced. It is then immediate to check that Q is propositionally satisfiable if and only if the resulting $\forall^{\pi}_{\mathbf{0}}$-formula φ_Q is satisfiable. Thus the NP-hardness of the satisfiability of $\forall^{\pi}_{\mathbf{0}}$-formulae follows immediately.

Having shown a lower bound for the s.p. for $\forall^{\pi}_{\mathbf{0}}$-formulae, we next give an upper bound for it, proving that it is in the NExpTime class and, furthermore, when restricted to a certain useful collection of $\forall^{\pi}_{\mathbf{0}}$-formulae, it is NP-complete.

As proved in Theorem 3, satisfiability of a $\forall_{\mathbf{0}}^{\pi}$-formula φ can be tested by first guessing a skeletal representation $\mathcal{S}$, whose size is polynomial in the size $|\varphi|$ of φ (since $|\mathit{Vars}(\mathcal{S})| < 3 \cdot |\mathit{Vars}(\varphi)|$), and then verify that the formula φ is modeled correctly by the realization $\mathbf{R}$ of S relative to (V, T), where $V = \mathit{Vars}(\varphi)$ and $T = \mathit{Vars}(\mathcal{S}) \setminus \mathit{Vars}(\varphi)$. Construction of the realization $\mathbf{R}$ takes polynomial time, however to verify that $\mathbf{R} \models \varphi$ can take exponential time. Indeed, it is easy to check that $\mathbf{R}$ models correctly φ if and only if it satisfies the *expansion* $\mathsf{Exp}_{\mathcal{S}}(\varphi)$ of φ relative to $\mathcal{S}$, which we define shortly. For a simple prenex $\forall_{\mathbf{0}}^{\pi}$-formula ψ, we put

$$\mathsf{exp}_{\mathcal{S}}(\psi) =_{\text{Def}} \begin{cases} \psi & \text{if } \psi \text{ is quantifier-free,} \\ \bigwedge_{\text{“}x' \in \bar{\pi}(y)\text{”} \in \mathcal{S}} \mathsf{exp}_{\mathcal{S}}(\chi^{x}_{x'}) & \text{if } \psi = (\forall x \in \bar{\pi}(y))\chi, \\ \bigwedge_{\text{“}[x',y'] \in z\text{”} \in \mathcal{S}} \mathsf{exp}_{\mathcal{S}}(\chi^{x,\,y}_{x',\,y'}) & \text{if } \psi = (\forall [x,y] \in z)\chi. \end{cases}$$

Then we put

$$\mathsf{Exp}_{\mathcal{S}}(\varphi) =_{\text{Def}} \mathsf{exp}_{\mathcal{S}}(\varphi_1) \wedge \ldots \wedge \mathsf{exp}_{\mathcal{S}}(\varphi_n),$$

where $\varphi_1, \ldots, \varphi_n$ are the (simple prenex) conjuncts of φ. If ℓ is the longest quantifier prefix of the formulae $\varphi_1, \ldots, \varphi_n$, then it turns out that $|\mathsf{Exp}_{\mathcal{S}}(\varphi)| = \mathcal{O}(|\varphi|^{2\ell}) = \mathcal{O}(|\varphi|^{2 \cdot |\varphi|})$, and therefore to test whether $\mathbf{R} \models \mathsf{Exp}_{\mathcal{S}}(\varphi)$ takes at most exponential time, showing that the s.p. for $\forall_{\mathbf{0}}^{\pi}$-formula is in NExpTime.

However, the same proof shows that if we restrict to the collection of $\forall_{\mathbf{0}}^{\pi}$-formulae whose quantifier prefixes are bounded by a constant $h \geq 0$, which we call $(\forall_0^{\pi})^{\leq h}$, then $|\mathsf{Exp}_{\mathcal{S}}(\varphi)|$ is only polynomial in $|\varphi|$, for any $(\forall_0^{\pi})^{\leq h}$-formula φ, and therefore to test whether $\mathbf{R}$ models correctly $\mathsf{Exp}_{\mathcal{S}}(\varphi)$, and in turn to test whether $\mathbf{R} \models \varphi$, takes polynomial time in $|\varphi|$, proving the following result:

▶ **Corollary 4.** *The s.p. for $(\forall_0^{\pi})^{\leq h}$-formulae is* NP*-complete, for any $h \geq 0$.* ◀

In the rest of the paper we describe some applications of $\forall_{\mathbf{0}}^{\pi}$-formulae in the field of knowledge representation. More specifically, in the next section we introduce a novel description logic whose consistency problem can be reduced to the s.p. for $(\forall_0^{\pi})^{\leq 2}$-formulae. Such description logic will then be extended with Horn-style rules in Section 5.

4 The description logic $\mathcal{DL}\langle\forall_{\mathbf{0}}^{\pi}\rangle$

Description logics are a family of logic-based formalisms which allow to represent knowledge about a domain of interest in terms of *concepts* (which denote sets of elements), *roles* (which represent relations between elements), and *individuals* (which denote domain elements). Each language in this family is mainly characterized by its set of *constructors*, which allow to form complex terms starting from *concept names*, *role names*, and *individual names* (see Table 1 for the syntax and semantics of the most widely used description logic constructs). A description logic *knowledge base* is a finite set of statements which define constraints on the domain structure.

Description logic semantics[4] is given in terms of *interpretations*. An interpretation $\mathcal{I}$ consists of a nonempty *domain* $\Delta^{\mathcal{I}}$ and an interpretation function assigning to each concept

[4] Here we are recalling the *descriptive* semantics. There are several other semantics that are out of the scope of this paper.

name a subset of $\Delta^{\mathcal{I}}$, to every role name a relation over $\Delta^{\mathcal{I}}$, and to every individual name a domain item in $\Delta^{\mathcal{I}}$. An interpretation $\mathcal{I}$ extends recursively to complex terms. An interpretation $\mathcal{I}$ that satisfies all the constraints of a knowledge base $\mathcal{K}$ is said to be a *model* for $\mathcal{K}$. A knowledge base is said to be *consistent* if it admits a model. Thus the *consistency problem* for description logic knowledge bases is to determine whether a knowledge base is consistent or not.

It turns out that the semantical definitions of several description logic statements Σ may be expressed as formulae of the form

$$\mathcal{I} \models \Sigma \text{ iff } \big(\forall x_1 \in \Delta^{\mathcal{I}}\big) \ldots \big(\forall x_n \in \Delta^{\mathcal{I}}\big)\, \Gamma_\Sigma,$$

where Γ_Σ is a Boolean combination of expressions of the types

$$u \in C^{\mathcal{I}},\ [u, u'] \in R^{\mathcal{I}},\ u = a^{\mathcal{I}},\ u = u',$$

with C, R, a respectively a concept term, a role term, and an individual name, and with u, u' ranging over the variables $x_1, \ldots, x_n$ (see Table 1).

This holds in particular for all the knowledge base statements allowed in the novel description logic $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$ defined next.

▶ **Definition 5.** Let $\mathcal{N}^c, \mathcal{N}^r, \mathcal{N}^i$ be the three denumerable, infinite and mutually disjoint collections of, respectively, concept, role, and individual names. $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-concept terms and $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-role terms are formed according to the following syntax rules:

$$\begin{array}{rcl} C, D & \longrightarrow & A \mid \top \mid \bot \mid \neg C \mid C \sqcup D \mid C \sqcap D \mid \{a\} \mid \exists R.\mathsf{Self} \mid \exists R.\{a\} \\ R, S & \longrightarrow & P \mid \mathbb{U} \mid R^- \mid \neg R \mid R \sqcup S \mid R \sqcap S \mid R_{C|} \mid R_{|D} \mid R_{C|D} \mid \mathsf{id}(C) \mid \mathsf{sym}(R) \end{array}$$

where C, D denote $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-concept terms, R, S denote $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-role terms, A, P denote a concept and a role name, respectively, and a denotes an individual name. A $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-knowledge base is then a finite collection of statements of the following types:

$$\begin{array}{lllll} C \equiv D\,, & C \sqsubseteq D\,, & R \equiv S\,, & R \sqsubseteq S\,, & C \sqsubseteq \forall R.D\,, \\ \exists R.C \sqsubseteq D\,, & R \circ R' \sqsubseteq S\,, & \mathsf{Trans}(R)\,, & \mathsf{Ref}(R)\,, & \mathsf{ASym}(R) \end{array}$$

where C, D are $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-concept terms and R, S, R' are $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-role terms.

Notice that the above definition of $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$ is not minimal, as we intended to give a clear and immediate overview of its expressive power.

The major limitation of $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$ (with respect to other description logics) is that value restriction and existential quantification are restricted to the left-hand side and right-hand side of inclusions, respectively. Moreover, number restrictions are not allowed. On the other hand, the set of allowed constructs is extremely large. In particular, complex role constructors can be used freely, in contrast with most expressive description logics. Additionally, reasoning in $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$ is NP-complete, as will be proved in the following theorem.

▶ **Theorem 6.** *The consistency problem for $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-knowledge bases is* NP*-complete.*

Proof. We will show that the consistency problem for $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-knowledge bases reduces to the satisfiability problem for $(\forall^{\pi}_0)^{\leq 2}$-formulae.

We begin with observing that we can restrict our attention to $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-knowledge bases containing only statements of the following types:

$$\begin{array}{llllllll} A \equiv \top\,, & A \equiv \neg B\,, & A \equiv B \sqcup B'\,, & A \equiv \{a\}\,, & A \sqsubseteq \forall P.B\,, & \exists P.A \sqsubseteq B\,, & A \equiv \exists P.\{a\}, \\ P \equiv \mathbb{U}\,, & P \equiv \neg Q\,, & P \equiv Q \sqcup Q'\,, & P \equiv Q^-\,, & P \equiv \mathsf{id}(A)\,, & P \equiv Q_{A|}\,, & P \circ P' \sqsubseteq Q, \\ \mathsf{Ref}(P) & & & & & & & \end{array}$$

$A^{\mathcal{I}}$	$\subseteq$	$\Delta^{\mathcal{I}}$	(concept name)
$P^{\mathcal{I}}$	$\subseteq$	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	(role name)
$a^{\mathcal{I}}$	$\in$	$\Delta^{\mathcal{I}}$	(individual name)
$\top^{\mathcal{I}}$	$=$	$\Delta^{\mathcal{I}}$	(universal concept)
$\bot^{\mathcal{I}}$	$=$	$\emptyset$	(bottom concept)
$(\neg C)^{\mathcal{I}}$	$=$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	(concept negation)
$(C \sqcup D)^{\mathcal{I}}$	$=$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	(concept union)
$(C \sqcap D)^{\mathcal{I}}$	$=$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	(concept intersection)
$\{a\}^{\mathcal{I}}$	$=$	$\{a^{\mathcal{I}}\}$	(nominal)
$(\exists R.\mathsf{Self})^{\mathcal{I}}$	$=$	$\{x \in \Delta^{\mathcal{I}} : [x,x] \in R^{\mathcal{I}}\}$	(self restriction)
$(\forall R.C)^{\mathcal{I}}$	$=$	$\{x \in \Delta^{\mathcal{I}} : (\forall [x,y] \in R^{\mathcal{I}})(y \in C^{\mathcal{I}})\}$	(value restriction)
$(\exists R.C)^{\mathcal{I}}$	$=$	$\{x \in \Delta^{\mathcal{I}} : (\exists y \in C^{\mathcal{I}})([x,y] \in R^{\mathcal{I}})\}$	(existential quantifier)
$(\leq nR.C)^{\mathcal{I}}$	$=$	$\{x \in \Delta^{\mathcal{I}} : \lvert\{y \in C^{\mathcal{I}} : [x,y] \in R^{\mathcal{I}}\}\rvert \leq n\}$	(number restrictions)
$(\geq nR.C)^{\mathcal{I}}$	$=$	$\{x \in \Delta^{\mathcal{I}} : \lvert\{y \in C^{\mathcal{I}} : [x,y] \in R^{\mathcal{I}}\}\rvert \geq n\}$	
$(R \subseteq S)^{\mathcal{I}}$	$=$	$\{x \in \Delta^{\mathcal{I}} : (\forall y \in \Delta^{\mathcal{I}})([x,y] \in R^{\mathcal{I}} \rightarrow [x,y] \in S^{\mathcal{I}})\}$	(role-value-map)
$\mathbb{U}^{\mathcal{I}}$	$=$	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	(universal role)
$(\neg R)^{\mathcal{I}}$	$=$	$(\Delta \times \Delta) \setminus R^{\mathcal{I}}$	(role negation)
$(R \sqcup S)^{\mathcal{I}}$	$=$	$R^{\mathcal{I}} \cup S^{\mathcal{I}}$	(role union)
$(R \sqcap S)^{\mathcal{I}}$	$=$	$R^{\mathcal{I}} \cap S^{\mathcal{I}}$	(role intersection)
$(R^{-})^{\mathcal{I}}$	$=$	$\{[x,y] \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} : [y,x] \in R^{\mathcal{I}}\}$	(role inverse)
$(R_{C\lvert})^{\mathcal{I}}$	$=$	$\{[x,y] \in R^{\mathcal{I}} : x \in C^{\mathcal{I}}\}$	(role restrictions)
$(R_{\lvert D})^{\mathcal{I}}$	$=$	$\{[x,y] \in R^{\mathcal{I}} : y \in D^{\mathcal{I}}\}$	
$(R_{C\lvert D})^{\mathcal{I}}$	$=$	$(R_{C\lvert})^{\mathcal{I}} \cap (R_{\lvert D})^{\mathcal{I}}$	
$id(C)^{\mathcal{I}}$	$=$	$\{[x,x] : x \in C^{\mathcal{I}}\}$	(role identity)
$(R \circ S)^{\mathcal{I}}$	$=$	$R^{\mathcal{I}} \circ S^{\mathcal{I}}$	(role composition)
$(R^{*})^{\mathcal{I}}$	$=$	$(R^{\mathcal{I}})^{*}$	(transitive closure)
$(\mathsf{sym}(R))^{\mathcal{I}}$	$=$	$R^{\mathcal{I}} \cup (R^{-})^{\mathcal{I}}$	(symmetric closure)
$\mathcal{I} \models C \sqsubseteq D$	$\Longleftrightarrow$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$	(inclusion axioms)
$\mathcal{I} \models R \sqsubseteq S$	$\Longleftrightarrow$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$	
$\mathcal{I} \models C \equiv D$	$\Longleftrightarrow$	$C^{\mathcal{I}} = D^{\mathcal{I}}$	(equivalence axioms)
$\mathcal{I} \models R \equiv S$	$\Longleftrightarrow$	$R^{\mathcal{I}} = S^{\mathcal{I}}$	
$\mathcal{I} \models \mathsf{Trans}(R)$	$\Longleftrightarrow$	$R^{\mathcal{I}} \circ R^{\mathcal{I}} \subseteq R^{\mathcal{I}}$	(role transitivity)
$\mathcal{I} \models \mathsf{Ref}(R)$	$\Longleftrightarrow$	$(\mathsf{id}(\exists R.\top))^{\mathcal{I}} \subseteq R^{\mathcal{I}}$	(role reflexivity)
$\mathcal{I} \models \mathsf{ASym}(R)$	$\Longleftrightarrow$	$R^{\mathcal{I}} \cap (R^{-})^{\mathcal{I}} = \emptyset$	(role asymmetry)
$\mathcal{I} \models C(a)$	$\Longleftrightarrow$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$	(concept assertion)
$\mathcal{I} \models R(a,b)$	$\Longleftrightarrow$	$[a^{\mathcal{I}}, b^{\mathcal{I}}] \in R^{\mathcal{I}}$	(role assertion)

Table 1 Description logic constructs

where A, B, B' are concept names, P, P', Q, Q' are role names, and a is an individual name, since any $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-knowledge base $\mathcal{K}$ can be easily transformed into a knowledge base $\mathcal{K}'$ which contains only statements of these types, and such that $\mathcal{K}$ is consistent if and only if $\mathcal{K}'$ is.

Next, we define a mapping τ from $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-statements to simple prenex $\forall^{\pi}_{\mathbf{0}}$-formulae as follows:

$$
\begin{array}{rcl}
\tau(A \equiv \top) & =_{\text{Def}} & (\forall x \in \bar{\pi}(\Delta))\,(x \in \bar{\pi}(A)) \\
\tau(A \equiv \neg B) & =_{\text{Def}} & (\forall x \in \bar{\pi}(\Delta))\,(x \in \bar{\pi}(A) \leftrightarrow x \notin \bar{\pi}(B)) \\
\tau(A \equiv B \sqcup B') & =_{\text{Def}} & (\forall x \in \bar{\pi}(\Delta))\,(x \in \bar{\pi}(A) \leftrightarrow x \in \bar{\pi}(B) \vee x \in \bar{\pi}(B')) \\
\tau(A \equiv \{a\}) & =_{\text{Def}} & (\forall x \in \bar{\pi}(\Delta))\,(x \in \bar{\pi}(A) \leftrightarrow x = a) \wedge a \in \bar{\pi}(A) \\
\tau(A \sqsubseteq \forall P.B) & =_{\text{Def}} & (\forall [x,y] \in P)\,(x \in \bar{\pi}(A) \rightarrow y \in \bar{\pi}(B)) \\
\tau(\exists P.A \sqsubseteq B) & =_{\text{Def}} & (\forall [x,y] \in P)\,(y \in \bar{\pi}(A) \rightarrow x \in \bar{\pi}(B)) \\
\tau(A \equiv \exists P.\{a\}) & =_{\text{Def}} & (\forall x \in \bar{\pi}(\Delta))\,(x \in \bar{\pi}(A) \leftrightarrow [x,a] \in P) \\
\tau(P \equiv \mathbb{U}) & =_{\text{Def}} & (\forall [x,y] \in \Delta)\,([x,y] \in P) \\
\tau(P \equiv \neg Q) & =_{\text{Def}} & (\forall x,y \in \bar{\pi}(\Delta))\,([x,y] \in P \leftrightarrow [x,y] \notin Q) \\
\tau(P \equiv Q \sqcup Q') & =_{\text{Def}} & (\forall x,y \in \bar{\pi}(\Delta))\,([x,y] \in P \leftrightarrow [x,y] \in Q \vee [x,y] \in Q') \\
\tau(P \equiv Q^{-}) & =_{\text{Def}} & (\forall x,y \in \bar{\pi}(\Delta))\,([x,y] \in P \leftrightarrow [y,x] \in Q) \\
\tau(P \equiv Q_{A|}) & =_{\text{Def}} & (\forall x,y \in \bar{\pi}(\Delta))\,([x,y] \in P \leftrightarrow [x,y] \in Q \wedge x \in \bar{\pi}(A)) \\
\tau(P \equiv \mathsf{id}(A)) & =_{\text{Def}} & (\forall x,y \in \bar{\pi}(\Delta))\,([x,y] \in P \leftrightarrow x = y \wedge x \in \bar{\pi}(A)) \\
\tau(P \circ P' \sqsubseteq Q) & =_{\text{Def}} & (\forall [x,y] \in P)\,(\forall [y',z] \in P')\,(y = y' \rightarrow [x,z] \in Q) \\
\tau(\mathsf{Ref}(P)) & =_{\text{Def}} & (\forall [x,y] \in P)\,([x,x] \in P)
\end{array}
$$

We remark that in the above definition of the mapping τ we are assuming that the collection *Vars* of the variables of the language $\forall^{\pi}_{\mathbf{0}}$ contains all the concept, role, and individual names. Moreover, we used the same symbol Δ which is also used to denote the domain of a description logic interpretation, under the assumption that $\Delta \notin \mathcal{N}^c \cup \mathcal{N}^r \cup \mathcal{N}^i$. These are just technical assumptions (not strictly necessary for the proof) which have been just introduced to enhance readability of the formulae $\tau(\cdot)$ and to emphasize the strong correlation between the semantical definitions of $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-statements and their corresponding $\forall^{\pi}_{\mathbf{0}}$-formulae.

Now let $\mathcal{K}$ be a $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-knowledge base. We define the $\forall^{\pi}_{\mathbf{0}}$-formula φ, expressing the consistency of $\mathcal{K}$, as follows

$$
\begin{array}{rcl}
\varphi & =_{\text{Def}} & \varphi_{\Delta} \wedge \varphi_{C} \wedge \varphi_{R} \wedge \varphi_{I} \wedge \varphi_{\mathcal{K}} \\
\varphi_{\Delta} & =_{\text{Def}} & (\forall [x,y] \in \Delta)\,([x,y] \notin \Delta) \\
\varphi_{C} & =_{\text{Def}} & \bigwedge\limits_{A \in \mathsf{Cpts}} ((\forall x \in \bar{\pi}(A))\,(x \in \bar{\pi}(\Delta)) \wedge (\forall [x,y] \in A)\,([x,y] \notin A)) \\
\varphi_{R} & =_{\text{Def}} & \bigwedge\limits_{P \in \mathsf{Rls}} ((\forall x \in \bar{\pi}(P))\,(x \notin \bar{\pi}(P)) \wedge (\forall [x,y] \in P)\,(x \in \bar{\pi}(\Delta) \wedge y \in \bar{\pi}(\Delta))) \\
\varphi_{I} & =_{\text{Def}} & \bigwedge\limits_{a \in \mathsf{Inds}} a \in \bar{\pi}(\Delta) \\
\varphi_{\mathcal{K}} & =_{\text{Def}} & \bigwedge\limits_{\Sigma \in \mathcal{K}} \tau(\Sigma)
\end{array}
$$

where $\mathsf{Cpts}, \mathsf{Rls}$, and Inds are respectively the sets of concept, role and individual names occurring in $\mathcal{K}$.

The consistency problem for $\mathcal{K}$ is equivalent to the satisfiability of φ, as we prove next.

Plainly, $\varphi_{\Delta}, \varphi_{C}, \varphi_{R}$, and φ_{I} guarantee that each model of φ can be easily turned into a $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-interpretation. Additionally, $\varphi_{\mathcal{K}}$ ensures that the $\mathcal{DL}\langle\forall^{\pi}_{\mathbf{0}}\rangle$-interpretation obtained in this way satisfies all the statements in $\mathcal{K}$.

Conversely, let $\mathcal{I}$ be a model for $\mathcal{K}$. Without loss of generality, we may assume that $\Delta^{\mathcal{I}}$ is a set belonging to the von Neumann hierarchy $\mathcal{V}$ (otherwise, we embed $\Delta^{\mathcal{I}}$ in $\mathcal{V}$). Let

$\mathbf{I} = (M_{\mathbf{I}}, \pi_{\mathbf{I}})$ be the $\forall_{\mathbf{0}}^{\pi}$-interpretation, induced by $\mathcal{I}$, defined by

$$\begin{array}{rll} \pi_{\mathbf{I}}(u,v) & =_{\text{Def}} & \{u, \{u,v\}, \Delta^{\mathcal{I}}\} \qquad \text{for all } u,v \in \mathcal{V} \\ M_{\mathbf{I}}\Delta & =_{\text{Def}} & \Delta^{\mathcal{I}} \\ M_{\mathbf{I}}A & =_{\text{Def}} & A^{\mathcal{I}} \qquad \text{for all } A \in \mathcal{N}^c \\ M_{\mathbf{I}}P & =_{\text{Def}} & \{\pi_{\mathbf{I}}(u,v) : [u,v] \in P^{\mathcal{I}}\} \qquad \text{for all } P \in \mathcal{N}^r \\ M_{\mathbf{I}}a & =_{\text{Def}} & a^{\mathcal{I}} \qquad \text{for all } a \in \mathcal{N}^i. \end{array}$$

Since $\mathbf{I}\Delta \in \pi_{\mathbf{I}}(u,v)$ for all $u,v \in \mathcal{V}$, from the well-foundedness of the membership relation it follows that $\mathbf{I}\Delta$ does not contain any pair (with respect to $\pi_{\mathbf{I}}$). Thus $x \in \mathbf{I}\bar{\pi}(A) \iff x \in A^{\mathcal{I}}$ and $\pi_{\mathbf{I}}(x,y) \in \mathbf{I}P \iff [x,y] \in P^{\mathcal{I}}$ follow from the definition of $\mathbf{I}$, and then $\mathbf{I}\tau(\Sigma) = \mathbf{true}$ if and only if $\mathcal{I}$ satisfies Σ, for all the statements $\Sigma \in \mathcal{K}$.

We conclude the proof by observing that each conjunct in φ contains at most two quantifiers (i.e., φ is a formula of $(\forall_0^{\pi})^{\leq 2}$), thus in view of Corollary 4 the satisfiability of φ can be checked in nondeterministic polynomial time, while the NP-hardness of this problem follows directly from the NP-completeness of the satisfiability problem for propositional formulae. ◀

5 Extending $\mathcal{DL}\langle\forall_{\mathbf{0}}^{\pi}\rangle$ with SWRL rules

In order to increase the expressive power of description logics, in [12] it was proposed to extend this framework with a simple form of Horn-style rules called SWRL rules. SWRL rules have the form

$$H \to B_1 \wedge \ldots \wedge B_n$$

where $H, B_1, \ldots, B_n$ are *atoms* of the forms $A(x), P(x,y), x = y, x \neq y$, with A a concept name, P a role name, and x, y either SWRL-variables or individual names.

A *binding* $\mathcal{B}(\mathcal{I})$ is any extension of the interpretation $\mathcal{I}$ which assigns a domain item to each SWRL-variable. An interpretation $\mathcal{I}$ satisfies a rule $H \to B_1 \wedge \ldots \wedge B_n$ if each binding $\mathcal{B}(\mathcal{I})$ which satisfies all the atoms $B_1, \ldots, B_n$ satisfies H also.

A $\mathcal{DL}\langle\forall_{\mathbf{0}}^{\pi}\rangle$-knowledge base $\mathcal{K}$ extended with a finite set of SWRL rules $\mathcal{R}$ is said to be *satisfiable* if and only if it has a model which satisfies all the rules in $\mathcal{R}$.

The reduction provided in Section 4 can be easily extended to cope with $\mathcal{DL}\langle\forall_{\mathbf{0}}^{\pi}\rangle$-knowledge bases extended with finite sets of SWRL rules, as shown in the following theorem.

▶ **Theorem 7.** *The consistency problem for $\mathcal{DL}\langle\forall_{\mathbf{0}}^{\pi}\rangle$-knowledge bases extended with finite sets of SWRL rules is decidable.*

Proof. Let $\mathcal{K}$ be a $\mathcal{DL}\langle\forall_{\mathbf{0}}^{\pi}\rangle$-knowledge base, and let $\mathcal{R}$ be a finite set of SWRL rules. Let us extend the mapping τ, defined in Theorem 6, to SWRL rules and atoms as follows:

$$\begin{array}{rll} \tau(H \to B_1 \wedge \ldots \wedge B_n) & =_{\text{Def}} & (\forall x_1, \ldots, x_m \in \bar{\pi}(\Delta))\,(\tau(H) \to \tau(B_1) \wedge \ldots \wedge \tau(B_n)) \\ \tau(A(x)) & =_{\text{Def}} & x \in \bar{\pi}(A) \\ \tau(P(x,y)) & =_{\text{Def}} & [x,y] \in P \\ \tau(x = y) & =_{\text{Def}} & x = y \\ \tau(x \neq y) & =_{\text{Def}} & x \neq y \end{array}$$

where $H, B_1, \ldots B_n$ are SWRL atoms, $x_1, \ldots, x_m$ are the SWRL variables occurring in $H \to B_1 \wedge \ldots \wedge B_n$, x, y can be either SWRL variables or individual names, and A, P are respectively a concept and a role name.

We conclude the proof by observing that the following $\forall_{\mathbf{0}}^{\pi}$-formula φ' is satisfiable if and only if the knowledge base $\mathcal{K}$ extended with $\mathcal{R}$ is consistent:

$$\varphi' =_{\text{Def}} \bigwedge_{\rho \in \mathcal{R}} \tau(\rho) \wedge \varphi,$$

where φ is built from $\mathcal{K}$ as described in Theorem 6, extending Cpts, Rls and Inds with the concept, role and individual names occurring in $\mathcal{R}$, respectively. ◀

6 Conclusions and future works

We have introduced the collection of quantified $\forall_{\mathbf{0}}^{\pi}$-formulae of set theory, which allow the explicit manipulation of ordered pairs, and proved that they have a decidable satisfiability problem. In fact, when restricted to $\forall_{\mathbf{0}}^{\pi}$-formulae whose conjuncts have quantifier prefixes of length bounded by a constant, the satisfiability problem is NP-complete.

In addition, we have introduced the novel description logic $\mathcal{DL}\langle\forall_{\mathbf{0}}^{\pi}\rangle$ and shown that its consistency check is NP-complete, since it can be reduced to the satisfiability test for a $\forall_{\mathbf{0}}^{\pi}$-formula whose conjuncts involve at most two quantifiers. Finally we have extended the description logic $\mathcal{DL}\langle\forall_{\mathbf{0}}^{\pi}\rangle$ with SWRL rules without disrupting the decidability of the knowledge base consistency problem.

In contrast with description logics, the semantics of set theory is *multi-level*, so that sets (and consequently relations) can be nested arbitrarily. In the light of this observation, we intend to investigate whether the description logic $\mathcal{DL}\langle\forall_{\mathbf{0}}^{\pi}\rangle$ can be extended with *meta-modeling* features (cf. [15]), which would allow to state relationships among elements of the conceptual model.

Finally, we intend to investigate if $\forall_{\mathbf{0}}^{\pi}$ (and consequently $\mathcal{DL}\langle\forall_{\mathbf{0}}^{\pi}\rangle$) can be extended with concrete domains, in order to promote definitively $\forall_{\mathbf{0}}^{\pi}$ as a language for knowledge representation, and, consequently, for the semantic web.

References

1 Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

2 Michael Breban, Alfredo Ferro, Eugenio G. Omodeo, and Jacob T. Schwartz. Decision procedures for elementary sublanguages of set theory. II. Formulas involving restricted quantifiers, together with ordinal, integer, map, and domain notions. *Communications on Pure and Applied Mathematics*, 34:177–195, 1981.

3 Domenico Cantone and Alfredo Ferro. *Techniques of computable set theory with applications to proof verification*, vol. XLVIII of *Comm. Pure Appl. Math.*, pages 901–945. Wiley, 1995.

4 Domenico Cantone, Alfredo Ferro, and Eugenio Omodeo. *Computable set theory*, vol. 6 of *Int'l Series of Monographs on Computer Science*. Oxford Science Publications. Clarendon Press, Oxford, UK, 1989.

5 Domenico Cantone, Cristiano Longo, and Marianna Nicolosi Asmundo. A Decision Procedure for a Two-sorted Extension of Multi-Level Syllogistic with the Cartesian Product and Some Map Constructs. In W. Faber and N Leone, (eds.), *25th Italian Conference on Computational Logic (CILC'10)*, 2010.

6 Domenico Cantone, Cristiano Longo, and Antonio Pisasale. Comparing Description Logics with Multi-level Syllogistics: the Description Logic $\mathcal{DL}\langle\mathsf{MLSS}^{\times}_{2,m}\rangle$. In *6th Workshop on Semantic Web Applications and Perspectives (SWAP)*, 2010.

7 Domenico Cantone, Eugenio Omodeo, and Alberto Policriti. *Set theory for computing: from decision procedures to declarative programming with sets*. Monographs in Computer Science. Springer-Verlag, New York, NY, USA, 2001.

8 Domenico Cantone, Eugenio G. Omodeo, Jacob T. Schwartz, and Pietro Ursino. Notes from the logbook of a proof-checker's project. In Nachum Dershowitz, editor, *Verification: Theory and Practice*, vol. 2772 of *LNCS*, pp. 182–207. Springer, 2003.

9 Domenico Cantone and Jacob T. Schwartz. Decision Procedures for Elementary Sublanguages of Set Theory: XI. Multilevel Syllogistic Extended by Some Elementary Map Constructs. *J. Autom. Reasoning*, 7(2):231–256, 1991.

10 Alfredo Ferro, Eugenio G. Omodeo, and Jacob T. Schwartz. *Decision Procedures for Elementary Sublanguages of Set Theory. I. Multi-level syllogistic and some extensions.*, volume XXXIII of *Comm. Pure Appl. Math.*, pages 599–608. Wiley, 1980.

11 Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The Even More Irresistible SROIQ. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2006)*, pages 57–67, AAAI Press, June 2006.

12 Ian Horrocks and Peter F. Patel-Schneider. A proposal for an OWL rules language. In S. I. Feldman, M. Uretsky, M. Najork, and C.E. Wills, (eds.), *WWW*, pp. 723–731. ACM, 2004.

13 Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Description Logic Rules. In M. Ghallab, C. D. Spyropoulos, N. Fakotakis, and N. M. Avouris, (eds.), *ECAI*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 80–84. IOS Press, 2008.

14 Carsten Lutz and Ulrike Sattler. The Complexity of Reasoning with Boolean Modal Logics. In Frank Wolter, Heinrich Wansing, Maarten de Rijke, and Michael Zakharyaschev, editors, *Advances in Modal Logic*, pages 329–348. World Scientific, 2000.

15 Boris Motik. On the Properties of Metamodeling in OWL. *J. Log. Comput.*, 17(4):617–637, 2007.

16 Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with rules. *J. Web Sem.*, 3(1):41–60, 2005.

17 Eugenio Omodeo, Domenico Cantone, and Alberto Policriti. Reasoning, Action and Interaction in AI Theories and Systems, Essays Dedicated to Luigia Carlucci Aiello. In O. Stock and M. Schaerf, (eds.), *Reasoning, Action and Interaction in AI Theories and Systems*, vol. 4155 of *LNCS*. Springer, 2006.

18 Franco Parlamento and Alberto Policriti. Undecidability results for restricted universally quantified formulae of set theory. *Comm. Pure Appl. Math.*, ILVI:57–73, 1993.

19 Jacob T. Schwartz, Domenico Cantone, and Eugenio G. Omodeo. *Computational Logic and Set Theory: Applying Formalized Logic to Analysis*. Texts in Computer Science. Springer-Verlag New York, USA, 2011.

20 Jacob T. Schwartz, Robert B. K. Dewar, Edmond Schonberg, and E Dubinsky. *Programming with sets; an introduction to SETL*. Springer-Verlag New York, USA, 1986.

Continuous Markovian Logic – From Complete Axiomatization to the Metric Space of Formulas*

Luca Cardelli[1], Kim G. Larsen[2], and Radu Mardare[2]

1 **Microsoft Research Cambridge**
7 J J Thomson Ave Cambridge CB3 0FB, UK
luca@microsoft.com

2 **Department of Computer Science, Aalborg University**
Selma Lagerlofs Vej 300, DK-9220 Aalborg, Denmark
{kgl,mardare}@cs.aau.dk

Abstract

Continuous Markovian Logic (CML) is a multimodal logic that expresses quantitative and qualitative properties of continuous-space and continuous-time labelled Markov processes (CMPs). The modalities of CML approximate the rates of the exponentially distributed random variables that characterize the duration of the labeled transitions. In this paper we present a sound and complete Hilbert-style axiomatization of CML for the CMP-semantics and prove some meta-properties including the small model property. CML characterizes stochastic bisimulation and supports the definition of a quantified extension of satisfiability relation that measures the compatibility of a model and a property. Relying on the small model property, we prove that this measure can be approximated, within a given error, by using a distance between logical formulas.

1998 ACM Subject Classification F.4.1 Modal logic; G.3 Markov processes

Keywords and phrases Probabilistic logic, axiomatization, Markov processes, metric semantics

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.144

1 Introduction

Many complex natural and man-made systems (e.g., biological, ecological, physical, social, financial, and computational) are modeled as stochastic processes in order to handle either a lack of knowledge or inherent randomness. These systems are frequently studied in interaction with discrete systems, such as controllers, or with interactive environments having continuous behavior. This context has motivated research aiming to develop a general theory of systems able to uniformly treat discrete, continuous and hybrid reactive systems. Two of the central questions of this research are *"when do two systems behave similarly up to some quantifiable observation error?"* and *"is there any (algorithmic) technique to check whether two systems have similar behaviours?"*. These questions are related to the problems of state space reduction (collapsing a model to an equivalent reduced model) and discretization (reduce a continuous or hybrid system to an equivalent discrete one), which are cornerstones in the field of stochastic systems.

In the case of probabilistic systems, *probabilistic bisimulation* [17] has been introduce to relate systems with identical probabilistic behaviours and probabilistic multimodal logic (PML) [16, 17, 1, 11, 13] has been used to characterize this equivalence: the logical equivalence

* This work was partially supported by Sapere Aude: DFF-Young Researchers Grant 10-085054 of the Danish Council for Independent Research.

induced by PML on probabilistic models coincides with probabilistic bisimulation [17, 20, 10]. However, in spite of the elegant theories supporting it, the concept of bisimulation remains too strict for applications. In modelling, the values of the parameters (rates or probabilities) are often approximated and consequently, one is interested to know whether two processes that differ by a small amount in real-valued parameters show similar (not necessarily identical) behaviours. In such cases, instead of bisimulation relation, one needs a metric to estimate the degree of similarity of two systems in terms of their behaviours.

For quantifying the behavioral similarity of probabilistic systems it has been introduced a class of pseudometrics [21, 6, 20]. In these settings, the distance between two processes is zero iff they are bisimilar; otherwise, they are closer when they differ by a small amount in their probabilistic behaviours. These pseudometrics can be defined on top of PML, as shown in [6, 20], by extending the satisfiability relation $P \Vdash \phi$ to a function d such that $d(P,\phi) \in [0,1]$ measures the "degree of satisfiability" between the process P and the property ϕ. The function d induces a distance D between processes by $D(P,P') = sup\{|d(P,\phi) - d(P',\phi)|, \phi \in \mathcal{L}\}$, where $\mathcal{L}$ is the set of logical formulas. However, the computability of D is sometimes problematic, as it is the computability of $d(P,\phi)$ for an infinite or extremely big process P and for this reason approximation techniques such as statistical model checking [15, 22] are used to evaluate $d(P,\phi)$ within a given error.

In this paper we develop and study the *continuous Markovian logic* (CML) which is similar to PML but developed for general stochastic (Markovian) systems. Our models are *continuous-time and continuous-space labelled Markovian processes* (CMPs) [10, 3, 4]. They generalize other probabilistic models such as *labeled Markov processes* [20, 9, 5, 8] and *Harsanyi type spaces* [12, 19]. CML contains modal operators indexed with transition labels a and positive rationals r. The formula $L^a_r\phi$ expresses the fact that the rate of the a-transitions from a given state to the set of states satisfying ϕ is *at least* r; similarly, $M^a_r\phi$ states that the rate is *at most* r.

In spite of their syntactic similarities, CML and PML are very different. While in the probabilistic case the two modal operators are dual, being related by the De Morgan duality $M^a_r\phi \leftrightarrow L^a_{1-r}\neg\phi$, in the stochastic case they are independent. Moreover, there exists no sound equivalence of type $\neg X^a_r\phi \leftrightarrow Y^a_s\neg\phi$ for $X,Y \in \{L,M\}$, hence no positive normal forms can be defined for CML formulas. This is because the rate of the transitions from a given state m to the set of states satisfying ϕ is not related to the rate of the transitions from m to the set of states satisfying $\neg\phi$. The differences are reflected in the sound-complete axiomatizations that we present both for CML and for its fragment without M^a_r-operators. Many axioms of PML, such as $\vdash L^a_r\top$ or $\vdash L^a_r\phi \to \neg L^a_s\neg\phi$ for $r+s<1$ from [23][1], are not sound for CMPs. Also at the level of the small model property, which in the case of PML relies on the fact that for a fixed integer q there exists a finite number of integers p such that $p/q \in [0,1]$ (see [23]), a series of nontrivial additional problems rise in the stochastic case.

The construction of a small model for a consistent CML-formula is the cornerstone of this paper supporting not only the weak completeness proofs, but also approximation techniques to evaluate the extension $d(P,\phi) \in [0,1]$ of satisfiability relation. In the context of a sound and complete axiomatization, one can turn the bisimulation-distance problem, which in the probabilistic case has been addressed semantically, into a syntactic problem centered on provability. Formally, the distance $\overline{d}(\phi,\psi) = sup\{|d(P,\phi) - d(P,\psi)|, P \in \mathfrak{P}\}$, where $\mathfrak{P}$ is the class of CMPs, measures the similarity between logical formulas in terms of provability: ϕ and ψ are close in $\overline{d}$ if they (or their negations) can be both proved from the same hypothesis.

[1] The semantics of [23] is in terms of systems where each action is enabled with probability 1.

In this context we prove the *strong robustness theorem*: $d(P,\phi) \leq d(P,\psi) + \overline{d}(\phi,\psi)$. In case that $\overline{d}$ is not computable or it is very expensive, one can use our finite model construction to approximate its value. Let $\widetilde{d}(\phi,\psi) = max\{|d(P,\phi) - d(P,\psi)|, P \in \Omega_p[\phi,\psi]\}$, where $\Omega_p[\phi,\psi]$ is the finite model (finite set of processes) constructed for $\phi \wedge \psi$ if it is consistent, or for $\neg(\phi \wedge \psi)$ otherwise, and $p \in \mathbb{N}$ is the parameter involved in the construction. This guarantees the *weak robustness theorem*: $d(P,\phi) \leq d(P,\psi) + \widetilde{d}(\phi,\psi) + 2/p$. Using this theorem, one can evaluate $d(P,\phi)$ from the value of $d(P,\psi)$ and this can be used, for instance, in the context of statistical model checking. Of course, the accuracy of this approximation depends on the similarity of ϕ and ψ from a provability perspective, which influences both the distance $\widetilde{d}(\phi,\psi)$ and the parameter p of the finite model construction.

To summarize, the achievements of this paper are as follows.

- We introduce Continuous Markovian Logic, a modal logic that expresses quantitative and qualitative properties of continuous Markov processes. CML is endowed with operators that approximate the labelled transition rates of CMPs and allows us to reason on approximated properties. This logic characterizes the stochastic bisimulation of CMPs.
- We present sound and complete Hilbert-style axiomatizations for CMP and for its M_r^a-free fragment. These are very different from the similar probabilistic cases, due to the structural differences between probabilistic and stochastic models and the differences are reflected by the axioms.
- We prove the finite model properties for CML and its restricted fragment. The construction of a finite model for a consistent formula is novel in the way it exploits the granularity and the Archimedian properties of positive rationals.
- We define a distance between logical formulas that corroborates with the distance between a model and a formula proposed in the literature for probabilistic systems. The organization of the space of logical formulas as a pseudometrizable space with a topology sensitive to provability is a novelty in the field of metric semantics. This structure guarantees the strong robustness theorem.
- We show that the complete axiomatization and the finite model construction can be used to approximate the syntactic distance $\overline{d}$. This idea opens new research perspectives on the direction of designing algorithms to estimate such distances within given errors.

The structure of the paper. The first section establishes some preliminary concepts and notations used in the paper. Section 3 introduces CMPs and their bisimulation. In Section 4 we define the logic CML and in Section 5 we present sound-complete axiomatizations for both CML and its M_r^a-free fragment proving, at the same time, the small model properties. Section 6 introduces the metric semantics and the results related to metrics and bisimulation. The paper also contains a conclusive section where we discuss new research directions deriving from this paper.

2 Preliminary definitions and notations

In this section we introduce some notations and establish the terminology used in the paper.

For arbitrary sets A, B, 2^A denotes the powerset of A and $[A \to B]$ the set of functions from A to B.

If (M,Σ) is a measurable space with σ-algebra $\Sigma \subseteq 2^M$, we use $\Delta(M,\Sigma)$ to denote the set of measures[2] $\mu : \Sigma \to \mathbb{R}^+$ on (M,Σ). We organize $\Delta(M,\Sigma)$ as a measurable space by

[2] Notice that in this paper we do not consider infinite rates.

considering the σ-algebra generated, for arbitrary $S \in \Sigma$ and $r > 0$, by the sets

$$\{\mu \in \Delta(M, \Sigma) : \mu(S) \geq r\}.$$

Given two measurable spaces (M, Σ) and (N, Θ), we use $[\![M \to N]\!]$ to denote the class of measurable mappings from (M, Σ) to (N, Θ).

Given a relation $\mathfrak{R} \subseteq M \times M$, the set $N \subseteq M$ is $\mathfrak{R}$-closed iff $\{m \in M \mid \exists n \in N, (n, m) \in \mathfrak{R}\} \subseteq N$. If (M, Σ) is a measurable space and $\mathfrak{R} \subseteq M \times M$, then $\Sigma(\mathfrak{R})$ denotes the set of measurable $\mathfrak{R}$-closed subsets of M.

3 Continuous Markov processes

Based on an equivalence between the definitions of Harsanyi type spaces [12, 19] and labelled Markov processes [20, 9, 5, 8] evidenced by Doberkat in the light of the Giry monad [7], we introduce the *continuous Markov processes* (CMPs). CMPs are models of stochastic systems with continuous state space and continuous time transitions. They are defined for a fixed countable set $\mathcal{A}$ of *transition labels* representing the types of interactions with the environment. If $a \in \mathcal{A}$, m is the current state of the system and N is a measurable set of states, the function $\theta(a)(m)$ is a measure on the state space and $\theta(a)(m)(N) \in \mathbb{R}^+$ represents the *rate* of an exponentially distributed random variable that characterizes the duration of an a-transition from m to arbitrary $n \in N$. Indeterminacy in such systems is resolved by races between events executing at different rates.

▶ **Definition 1** (Continuous Markov processes)**.** Given an analytic set (M, Σ), where Σ is the Borel algebra generated by the topology, an *$\mathcal{A}$-continuous Markov kernel* is a tuple $\mathcal{M} = (M, \Sigma, \theta)$, where $\theta : \mathcal{A} \to [\![M \to \Delta(M, \Sigma)]\!]$. M is the support set of $\mathcal{M}$ denoted by $supp(\mathcal{M})$. If $m \in M$, $(\mathcal{M}, m)$ is an *$\mathcal{A}$-continuous Markov process*.

Notice that $\theta(a)$ is a measurable mapping between (M, Σ) and $\Delta(M, \Sigma)$. This is equivalent with the conditions on the two-variable *rate function* used in [10] to define continuous Markov processes (for the proof of the equivalence see, e.g. Proposition 2.9, of [7]).

In the rest of the paper we assume that the set of transition labels $\mathcal{A}$ is fixed. We denote by $\mathfrak{M}$ the class of $\mathcal{A}$-continuous Markov kernels (CMKs) and we use $\mathcal{M}, \mathcal{M}_i, \mathcal{M}'$ to range over $\mathfrak{M}$. We denote by $\mathfrak{P}$ the set of $\mathcal{A}$-CMPs and we use P, P_i, P' to range over $\mathfrak{P}$.

The stochastic bisimulation for CMPs follows the line of Larsen-Skou probabilistic bisimulation [17, 8, 20].

▶ **Definition 2** (Stochastic Bisimulation)**.** Given $\mathcal{M} = (M, \Sigma, \theta) \in \mathfrak{M}$, a *rate-bisimulation relation* on $\mathcal{M}$ is a relation $\mathfrak{R} \subseteq M \times M$ such that $(m, n) \in \mathfrak{R}$ iff for any $C \in \Sigma(\mathfrak{R})$ and any $a \in \mathcal{A}$,

$$\theta(a)(m)(C) = \theta(a)(n)(C).$$

Two processes $(\mathcal{M}, m)$ and $(\mathcal{M}, n)$ are *stochastic bisimilar*, written $m \sim_{\mathcal{M}} n$, if they are related by a rate-bisimulation relation.

Observe that, for any $\mathcal{M} \in \mathfrak{M}$ there exist rate-bisimulation relations as, for instance, is the identity relation on $\mathcal{M}$; the stochastic bisimulation is the largest rate-bisimulation.

If $\mathcal{M} = (M, \Sigma, \theta), \mathcal{M}' = (M', \Sigma', \theta') \in \mathfrak{M}$, then $\mathcal{M}'' = (M'', \Sigma'', \theta'') = \mathcal{M} \uplus \mathcal{M}'$ if $M'' = M \uplus M'$, Σ'' is generated by $\Sigma \uplus \Sigma'$ and for any $a \in \mathcal{A}$, $N \in \Sigma$ and $N' \in \Sigma'$,

$$\theta''(a)(m)(N \uplus N') = \begin{cases} \theta(a)(m)(N) & \text{if } m \in M \\ \theta'(a)(m)(N') & \text{if } m \in M' \end{cases}$$

Notice that $\mathcal{M}'' \in \mathfrak{M}$. If $m \in M$ and $m' \in M'$, we say that $(\mathcal{M}, m)$ and $(\mathcal{M}', m')$ are *bisimilar* written $(\mathcal{M}, m) \sim (\mathcal{M}', m')$ whenever $m \sim_{\mathcal{M} \uplus \mathcal{M}'} m'$.

4 Continuous Markovian Logics

In this section we introduce the continuous Markovian logic (CML) for semantics based on CMPs. In addition to the Boolean operators, this logic is provided with *stochastic modal operators* that approximate the rates of transitions. For $a \in \mathcal{A}$ and $r \in \mathbb{Q}_+$, $L^a_r\phi$ characterizes $(\mathcal{M}, m)$ whenever the rate of the a-transition from m to the class of the states satisfying ϕ is *at least* r; symmetrically, $M^a_r\phi$ is satisfied when this rate is *at most* r. CMLs extends the probabilistic logics [1, 16, 13, 23, 11] to stochastic domains. The obvious structural similarities between the probabilistic and the stochastic models are not preserved when we consider the logic. By focusing on general measures instead of probabilistic measures in the definition of the transition systems, many of the axioms of probabilistic logics are not sound for stochastic semantics. This is the case, for instance, with $\vdash L^a_r\top$ or $\vdash L^a_r\phi \to \neg L^a_s\neg\phi$ for $r + s < 1$ which are proposed in [11]. Moreover, while in probabilistic settings the operators L^a_r and M^a_S are dual, satisfying $M^a_r\phi = L^a_{1-r}\neg\phi$, they became independent in stochastic semantics. For this reason, in the next section we study two CML logics with complete axiomatizations, $\mathcal{L}$ involving only the stochastic operators of type L^a_r and $\mathcal{L}^+$ that contains both L^a_r and M^a_s.

▶ **Definition 3** (Syntax)**.** Given a countable set $\mathcal{A}$, the formulas of $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}^+(\mathcal{A})$ respectively are introduced by the following grammars, for arbitrary $a \in \mathcal{A}$ and $r \in \mathbb{Q}_+$.

$$\mathcal{L}(\mathcal{A}): \quad \phi := \top \mid \neg\phi \mid \phi \wedge \phi \mid L^a_r\phi,$$

$$\mathcal{L}^+(\mathcal{A}): \quad \phi := \top \mid \neg\phi \mid \phi \wedge \phi \mid L^a_r\phi \mid M^a_r\phi.$$

In addition, we assume all the Boolean operators and $\bot = \neg\top$, as well as the derived operator $E^a_r\phi = L^a_r\phi \wedge M^a_r\phi$.

In what follows we use the same set $\mathcal{A}$ of labels used with CMPs. The semantics of $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}^+(\mathcal{A})$, called in this paper *Markovian semantics*, are defined by the *satisfiability relation* for arbitrary $\mathcal{A}$-CMPs $(\mathcal{M}, m)$ with $\mathcal{M} = (M, \Sigma, \theta) \in \mathfrak{M}$, by:

$\mathcal{M}, m \Vdash \top$ always,
$\mathcal{M}, m \Vdash \neg\phi$ iff it is not the case that $\mathcal{M}, m \Vdash \phi$,
$\mathcal{M}, m \Vdash \phi \wedge \psi$ iff $\mathcal{M}, m \Vdash \phi$ and $\mathcal{M}, m \Vdash \psi$,
$\mathcal{M}, m \Vdash L^a_r\phi$ iff $\theta(a)(m)([\![\phi]\!]_{\mathcal{M}}) \geq r$,
$\mathcal{M}, m \Vdash M^a_r\phi$ iff $\theta(a)(m)([\![\phi]\!]_{\mathcal{M}}) \leq r$,
where $[\![\phi]\!]_{\mathcal{M}} = \{m \in M | \mathcal{M}, m \Vdash \phi\}$.

When it is not the case that $\mathcal{M}, m \Vdash \phi$, we write $\mathcal{M}, m \nVdash \phi$.

We have that $\mathcal{M}, m \nVdash \bot$ always and that $\mathcal{M}, m \Vdash E^a_r\phi$ iff $\theta(a)(m)([\![\phi]\!]_{\mathcal{M}}) = r$. Notice that $E^a_r\phi$ characterizes the process that can do an a-transition to the set of processes satisfying ϕ with the rate r. So, in this case one can express the exact rate of the transitions. This is always possible in probabilistic logic where M^a_r and L^a_r are dual operators and consequently E^a_r is always definable. In the stochastic case L^a_r, M^a_r and E^a_r are mutually independent. We chose not to study a Markovian logic that involves only the E^a_r operators because in many applications we do not know the exact rates of the transitions and it is more useful to work with approximations such as M^a_r or L^a_r.

The semantics of $L^a_r\phi$ and $M^a_r\phi$ are well defined only if $[\![\phi]\!]_{\mathcal{M}}$ is measurable. This is guaranteed by the fact that $\theta(a)$ is a measurable mapping between (M, Σ) and $\Delta(M, \Sigma)$, as proved in the next lemma.

▶ **Lemma 4.** *For any $\phi \in \mathcal{L}^+(\mathcal{A})$ and any $\mathcal{M} = (M, \Sigma, \theta) \in \mathfrak{M}$, $[\![\phi]\!]_{\mathcal{M}} \in \Sigma$.*

Proof. Induction on ϕ: for $\phi = L^a_r\psi$, the inductive hypothesis guarantees that $[\![\psi]\!]_{\mathcal{M}} \in \Sigma$, hence, $\{\mu \in \Delta(M,\Sigma) | \mu([\![\psi]\!]_{\mathcal{M}}) \geq r\}$ is measurable in $\Delta(M,\Sigma)$. Because $\theta(a)$ is a measurable mapping, we obtain that $[\![L^a_r\psi]\!]_{\mathcal{M}} = (\theta(a))^{-1}(\{\mu \in \Delta(M,\Sigma) | \mu([\![\psi]\!]_{\mathcal{M}}) \geq r\})$ is measurable. Similarly it can be proved for $\phi = M^a_r\psi$. ◀

A formula ϕ is *satisfiable* if there exists $\mathcal{M} = (M,\Sigma,\theta) \in \mathfrak{M}$ and $m \in M$ such that $\mathcal{M}, m \Vdash \phi$. ϕ is *valid*, denoted by $\Vdash \phi$, if $\neg\phi$ is not satisfiable.

5 Complete axiomatizations

In this section we present two Hilbert-style axiomatizations, one for $\mathcal{L}(\mathcal{A})$ and one for $\mathcal{L}^+(\mathcal{A})$, and we prove that they are sound and (weak) complete against the Markovian semantics. Both axiomatizations, as in the case of the axiomatization proposed in [23] for probabilistic systems, contain infinitary rules that encode the Archimedean properties of $\mathbb{Q}_+ \cup \{+\infty\}$. However, as has been shown in [14] following the lines of [13], a finitary axiomatic system can be given at the price of replacing the stochastic operators with some more complex operators. For our purpose of reasoning on approximated properties of Markovian processes, a complete axiomatization involving only the stochastic operators (and their Archimedian rules) is more useful.

5.1 Axiomatization for $\mathcal{L}(\mathcal{A})$

Table 1 contains a Hilbert-style axiomatization for $\mathcal{L}(\mathcal{A})$. The axioms and rules, considered in addition to the axiomatization of classic propositional logic, are given for propositional variables $\phi, \psi \in \mathcal{L}(\mathcal{A})$, for arbitrary $a \in \mathcal{A}$ and $s, r \in \mathbb{Q}^+$.

(A1):	$\vdash L^a_0\phi$
(A2):	$\vdash L^a_{r+s}\phi \to L^a_r\phi$
(A3):	$\vdash L^a_r(\phi \wedge \psi) \wedge L^a_s(\phi \wedge \neg\psi) \to L^a_{r+s}\phi$
(A4):	$\vdash \neg L^a_r(\phi \wedge \psi) \wedge \neg L^a_s(\phi \wedge \neg\psi) \to \neg L^a_{r+s}\phi$
(R1):	If $\vdash \phi \to \psi$ then $\vdash L^a_r\phi \to L^a_r\psi$
(R2):	If $\forall r < s, \vdash \phi \to L^a_r\psi$ then $\vdash \phi \to L^a_s\psi$
(R3):	If $\forall r > s, \vdash \phi \to L^a_r\psi$ then $\vdash \phi \to \bot$

Table 1 The axiomatic system of $\mathcal{L}(\mathcal{A})$

This axiomatic system has some similarities to the axiomatic system of probabilistic logic proposed in [23] for Harsanyi type spaces. The main difference is that the axioms of probabilistic logic $\vdash L^a_r\top$ and $\vdash L^a_r\phi \to \neg L^a_s\neg\phi$ for $r + s \leq 1$ are not sound for the Markovian semantics and this changes the entire proof structure. We also have two Archimedean properties reflected in (R2) and (R3); while the first allows us to argue on convergent sequences of rationals, the second excludes the models with infinite rates.

As usual, we say that a formula ϕ is *provable*, denoted by $\vdash \phi$, if it can be proved from the given axioms and rules. We say that ϕ is *consistent*, if $\phi \to \bot$ is not provable. Given a set Φ of formulas, we say that Φ proves ϕ, $\Phi \vdash \phi$, if from the formulas of Φ and the axioms one can prove ϕ. Φ is *consistent* if it is not the case that $\Phi \vdash \bot$. For a sublanguage $\mathcal{L} \subseteq \mathcal{L}^+(\mathcal{A})$, we say that Φ is $\mathcal{L}$-*maximally consistent* if Φ is consistent and no formula of $\mathcal{L}$ can be added to it without making it inconsistent.

▶ **Theorem 5** (Soundness). *The axiomatic system of $\mathcal{L}(\mathcal{A})$ is sound for the Markovian semantics, i.e., for any $\phi \in \mathcal{L}(\mathcal{A})$, if $\vdash \phi$ then $\Vdash \phi$.*

In what follows we prove the finite model property for $\mathcal{L}(\mathcal{A})$ using the filtration method adapted for CMPs. This result will eventually establish the (weak) completeness of the axiomatic system for the Markovian semantics, meaning that everything that is true for all the models is also provable. This logic is not complete because the stochastic operators are not compact.

To prove the weak completeness we will construct, for an arbitrary consistent formula $\psi \in \mathcal{L}(\mathcal{A})$, a model $(\mathcal{M}_\psi, \Gamma)$ where $supp(\mathcal{M}_\psi)$ is a finite set of $\mathcal{L}(\mathcal{A})$-consistent sets of formulas. As usual with the filtration method, the key argument is the truth lemma: $\psi \in \Gamma$ iff $\mathcal{M}_\psi, \Gamma \Vdash \psi$. A similar construction has been proposed in [23] for probabilistic logic, where the finite model property derives from the fact that the number of rationals of type $\frac{p}{n}$, for a fixed integer n, is finite within $[0, 1]$. The same property does not hold in our case, as the focus is on $[0, \infty)$, and instead we need a more complicated construction.

Before proceeding with the construction, we fix some notations.

For $n \in \mathbb{N}$, $n \neq 0$, let $\mathbb{Q}_n = \{\frac{p}{n} : p \in \mathbb{N}\}$. If $S \subseteq \mathbb{Q}$ is finite, the *granularity of* S, $gr(S)$, is the least common multiple of the denominators of the elements of S.
The *modal depth of* $\phi \in \mathcal{L}(\mathcal{A})$ is defined by $md(\top) = 0$, $md(\neg\phi) = md(\phi)$, $md(\phi \wedge \psi) = max(md(\phi), md(\psi))$ and $md(L^a_r\phi) = md(\phi) + 1$.
The *granularity of* $\phi \in \mathcal{L}$ is $gr(\phi) = gr(R)$, where $R \subseteq \mathbb{Q}_+$ is the set of indexes r of the operators L^a_r present in ϕ; the *upper bound of* ϕ is $max(\phi) = max(R)$.
The *actions of* ϕ is the set $act(\phi) \subseteq \mathcal{A}$ of indexes $a \in \mathcal{A}$ of the operators L^a_r present in ϕ.
For arbitrary $n \in \mathbb{N}$ and $A \subseteq \mathcal{A}$, let $\mathcal{L}_n(A)$ be the sublanguage of $\mathcal{L}(\mathcal{A})$ that uses only modal operators L^a_r with $r \in \mathbb{Q}_n$ and $a \in A$. For $\Lambda \subseteq \mathcal{L}(\mathcal{A})$, let $[\Lambda]_n = \Lambda \cup \{\phi \in \mathcal{L}_n(\mathcal{A}) : \Lambda \vdash \phi\}$.

Consider a consistent formula $\psi \in \mathcal{L}(\mathcal{A})$ with $gr(\psi) = n$ and $act(\psi) = A$.
Let $\mathcal{L}[\psi] = \{\phi \in \mathcal{L}_n(A) \mid max(\phi) \leq max(\psi), md(\phi) \leq md(\psi)\}$.

In what follows we construct $\mathcal{M}_\psi \in \mathfrak{M}$ such that each $\Gamma \in supp(\mathcal{M}_\psi)$ is a consistent set of formulas that contains an $\mathcal{L}[\psi]$-maximally consistent set of formulas and each $\mathcal{L}[\psi]$-maximally consistent set is contained in some $\Gamma \in supp(\mathcal{M}_\psi)$. And we will prove that for $\phi \in \mathcal{L}[\psi]$, $\phi \in \Gamma$ iff $\mathcal{M}_\psi, \Gamma \Vdash \phi$.

Let $\Omega[\psi]$ be the set of $\mathcal{L}[\psi]$-maximally consistent sets of formulas. $\Omega[\psi]$ is finite and any $\Lambda \in \Omega[\psi]$ contains finitely many *nontrivial formulas*[3]; in the rest of this construction we only count non-trivial formulas while ignoring the rest and we use $\bigwedge \Lambda$ to denote the conjunction of the nontrivial formulas of Λ.

For each $\Lambda \in \Omega[\psi]$, such that $\{\phi_1, ..., \phi_i\} \subseteq \Lambda$ is its set of its non-trivial formulas, we construct $\Lambda^+ \supseteq [\Lambda]_n$ with the property that $\forall \phi \in \Lambda$ and $a \in \mathcal{A}$ there exists $\neg L^a_r\phi \in \Lambda^+$.

The construction step [ϕ_1 versus Λ:]
(R3) guarantees that $\exists r \in \mathbb{Q}_n$ s.t. $[\Lambda]_n \cup \{\neg L^a_r\phi_1\}$ is consistent (suppose that this is not the case, then $\vdash \bigwedge \Lambda \to L^a_r\phi_1$ for all $r \in \mathbb{Q}_n$ implying that $\bigwedge \Lambda$ inconsistent - impossible). Let $y^a_1 = min\{s \in \mathbb{Q}_n : [\Lambda]_n \cup \{\neg L^a_s\phi_1\}$ is consistent$\}$ and $x^a_1 = max\{s \in \mathbb{Q}_n : L^a_s\phi_1 \in [\Lambda]_n\}$ ((R3) guarantees the existence of max, because otherwise $\vdash \bigwedge \Lambda \to L^a_r\phi_1$ for all r implying $\bigwedge \Lambda$ inconsistent - impossible). (R2) implies that $\exists r \in \mathbb{Q} \setminus \mathbb{Q}_n$ s.t., $x^a_1 < r < y^a_1$ and $\{\neg L^a_r\phi_1\} \cup [\Lambda]_n$ is consistent (otherwise, $\vdash \bigwedge \Lambda \to L^a_r\phi_1$ for all $r < y^a_1$ and due to (R2), $\vdash \bigwedge \Lambda \to L^a_{y^a_1}\phi_1$ -

[3] By nontrivial formulas we mean the formulas that are not obtained from more basic consistent ones by boolean derivations. For instance $p \vee q \to p$, $p \wedge p$, $p \vee p$ are trivial formulas.

contradiction with the consistency of Λ). Obviously, $r \notin \mathbb{Q}_n$. Let $n_1 = gran\{1/n, r\}$. Let $s_1^a = min\{s \in \mathbb{Q}_{n_1} : [\Lambda]_{n_1} \cup \{\neg L_s^a \phi_1\} \text{ is consistent}\}$, $\Lambda_1^a = \Lambda \cup \{\neg L_{s_1^a}^a \phi_1\}$ and $\Lambda_1 = \bigcup_{a \in A} \Lambda_1^a$.

The construction step [ϕ_2 versus Λ_1:]
As before, let $y_2^a = min\{s \in \mathbb{Q}_{n_1} : [\Lambda_1]_{n_1} \cup \{\neg L_s^a \phi_2\} \text{ is consistent}\}$ and $x_2^a = max\{s \in \mathbb{Q}_{n_1} : L_s^a \phi_2 \in [\Lambda_1]_{n_1}\}$. There exists $r \in \mathbb{Q} \setminus \mathbb{Q}_{n_1}$ s.t., $x_2^a < r < y_2^a$ and $\{\neg L_r^a \phi_2\} \cup [\Lambda_1]_{n_1}$ is consistent. Let $n_2 = gran\{1/n_1, r\}$. Let $s_2^a = min\{s \in \mathbb{Q}_{n_2} : [\Lambda]_{n_2} \cup \{\neg L_s^a \phi_2\} \text{ is consistent}\}$, $\Lambda_2^a = \Lambda_1 \cup \{\neg L_{s_2^a}^a \phi_2\}$ and $\Lambda_2 = \bigcup_{a \in A} \Lambda_2^a$.

We repeat this construction step for $[\phi_3$ versus $\Lambda_2]$,..,$[\phi_i$ versus $\Lambda_{i-1}]$ and in a finite number of steps we eventually obtain $\Lambda \subseteq \Lambda_1 \subseteq ... \subseteq \Lambda_i$, where Λ_i is a consistent set containing a finite set of nontrivial formulas. Let $n_\Lambda = gran\{1/n_1, .., 1/n_i\}$. We make this construction for all $\Lambda \in \Omega[\psi]$. Let $p = gran\{1/n_\Lambda : \Lambda \in \Omega[\psi]\}$. Notice that $p > n$. Let $\Lambda^+ = [\Lambda_i]_p$ and $\Omega^+[\psi] = \{\Lambda^+ : \Lambda \in \Omega[\psi]\}$.

▶ Remark. Any consistent formula $\phi \in \mathcal{L}[\psi]$ is an element of a set $\Lambda^+ \in \Omega^+[\psi]$. For each $\Lambda \in \Omega[\psi]$, each $\phi \in \Lambda$ and $a \in \mathcal{A}$, there exist $s, t \in \mathbb{Q}_p$, $s < t$, such that $L_s^a \phi, \neg L_t^a \phi \in \Lambda^+$. Moreover, for any Λ^+ there exists a formula ρ such that $\phi \in \Lambda^+$ iff $\vdash \rho \to \phi$.

Let Ω_p be the set of $\mathcal{L}_p(A)$-maximally consistent sets of formulas. We fix an injective (choice) function $f : \Omega^+[\psi] \to \Omega_p$ such that for any $\Lambda^+ \in \Omega^+[\psi]$, $\Lambda^+ \subseteq f(\Lambda^+)$. We denote by $\Omega_p[\psi] = f(\Omega^+[\psi])$. For $\phi \in \mathcal{L}[\psi]$, let $[\![\phi]\!] = \{\Gamma \in \Omega_p[\psi] : \phi \in \Gamma\}$. Anticipating the further construction, we will use $\Omega_p[\psi]$ as the support-set for $\mathcal{M}_\psi$. For this reason we establish some properties for this set.

▶ **Lemma 6.** *1. $\Omega_p[\psi]$ is finite.*
2. $2^{\Omega_p[\psi]} = \{[\![\phi]\!] : \phi \in \mathcal{L}[\psi]\}$.
3. For any $\phi_1, \phi_2 \in \mathcal{L}[\psi]$, $\vdash \phi_1 \to \phi_2$ iff $[\![\phi_1]\!] \subseteq [\![\phi_2]\!]$.
4. For any $\Gamma \in \Omega_p[\psi]$, $\phi \in \mathcal{L}[\psi]$ and $a \in A$ there exist $x = max\{r \in \mathbb{Q}_p : L_r^a \phi \in \Gamma\}$, $y = min\{r \in \mathbb{Q}_p : \neg L_r^a \phi \in \Gamma\}$ and $y = x + 1/p$.

Proof. 4. $L_x^a \phi, \neg L_y^a \phi \in \Gamma$ implies $x \neq y$. If $x > y$, $L_x^a \phi \in \Gamma$ entails (Axiom (A2)) $L_y^a \phi \in \Gamma$, contradicting the consistency of Γ. If $x + 1/p < y$, then $L_{x+1/p}^a \phi \notin \Gamma$, i.e. $\neg L_{x+1/p}^a \phi \in \Gamma$ implying that $x + 1/p \geq y$ - contradiction. ◀

Let Ω be the set of $\mathcal{L}(\mathcal{A})$-maximally consistent sets of formulas. We fix an injective (choice) function $g : \Omega_p \to \Omega$ such that for any $\Gamma \in \Omega_p$, $\Gamma \subseteq \pi(\Gamma)$; we denote $g(\Gamma)$ by Γ^∞.

▶ **Lemma 7.** *For any $\Gamma \in \Omega_p[\psi]$, $\phi \in \mathcal{L}[\psi]$ and $a \in A$, there exists*

$$z = sup\{r \in \mathbb{Q} : L_r^a \phi \in \Gamma^\infty\} = inf\{r \in \mathbb{Q} : \neg L_r^a \phi \in \Gamma^\infty\} \text{ and } x \leq z < y.$$

Proof. Let $x^\infty = sup\{r \in \mathbb{Q} : L_r^a \phi \in \Gamma^\infty\}$ and $y^\infty = inf\{r \in \mathbb{Q} : \neg L_r^a \phi \in \Gamma^\infty\}$. Suppose that $x^\infty < y^\infty$. Then there exists $r \in \mathbb{Q}$ such that $x^\infty < r < y^\infty$. Hence, $\neg L_r^a \phi, L_r^a \phi \in \Gamma^\infty$ - impossible because Γ^∞ is consistent. Suppose that $x^\infty > y^\infty$. Then there exists $r \in \mathbb{Q}$ such that $x^\infty > r > y^\infty$. As Γ^∞ is maximally consistent we have either $L_r^a \phi \in \Gamma^\infty$ or $\neg L_r^a \phi \in \Gamma^\infty$. The first case contradicts the definition of x^∞ while the second the definition of y^∞. Hence, $x \leq z \leq y$. If $z = y$, then $L_z^a \phi, \neg L_z^a \phi \in \Gamma$ contradicting the consistency of Γ. ◀

We denote z by a_ϕ^Γ and now we can define $\mathcal{M}_\psi$.

▶ **Lemma 8.** *If $\theta_\psi : \mathcal{A} \to [\Omega_p[\psi] \to \Delta(\Omega_p[\psi], 2^{\Omega_p[\psi]})]$ is defined for arbitrary $a \in \mathcal{A}$, $\Gamma \in \Omega_q[\psi]$ and $\phi \in \mathcal{L}[\psi]$ by $\theta_\psi(a)(\Gamma)([\![\phi]\!]) = a_\phi^\Gamma$, then $\mathcal{M}_\psi = (\Omega_p[\psi], 2^{\Omega_p[\psi]}, \theta_\psi) \in \mathfrak{M}$.*

Proof. The central problem is to prove that for arbitrary $\Gamma \in \Omega_p[\psi]$ and $a \in A$, the function $\theta_\psi(a)(\Gamma) : 2^{\Omega_p[\psi]} \to \mathbb{R}^+$ is well defined and a measure on $(\Omega_p[\psi], 2^{\Omega_p[\psi]})$. Further, because the space is discrete with finite support, we obtain that $\theta_\psi(a) \in [\![\Omega_p[\psi] \to \Delta(\Omega_p[\psi], 2^{\Omega_p[\psi]})]\!]$.

Suppose that for $\phi_1, \phi_2 \in \mathcal{L}[\psi]$ we have $[\![\phi_1]\!] = [\![\phi_2]\!]$. Then, from Lemma 6, $\vdash \phi_1 \leftrightarrow \phi_2$ and $\vdash L^a_r\phi_1 \leftrightarrow L^a_r\phi_2$. Hence, $a^\Gamma_{\phi_1} = a^\Gamma_{\phi_2}$ proving that $\theta_\psi(a)(\Gamma)$ is well defined.

Now we prove that $\theta_\psi(a)(\Gamma)$ is a measure.
For showing $\theta_\psi(a)(\Gamma)(\emptyset) = 0$, we show that for any $r > 0$, $\vdash \neg L^a_r\bot$. This is sufficient, as (A1) guarantees that $\vdash L^a_0\bot$ and $[\![\bot]\!] = \emptyset$. Suppose that there exists $r > 0$ such that $L^a_r\bot$ is consistent. Let $\epsilon \in (0, r) \cap \mathbb{Q}$. Then (A2) gives $\vdash L^a_r\bot \to L^a_\epsilon\bot$. Hence, $\vdash L^a_r\bot \to (L^a_r(\bot \wedge \bot) \wedge L^a_\epsilon(\bot \wedge \neg\bot))$ and applying (A3), $\vdash L^a_r\bot \to L^a_{r+\epsilon}\bot$. Repeating this argument, we can prove that $\vdash L^a_r\bot \to L^a_s\bot$ for any s and (R3) confirms the inconsistency of $L^a_r\bot$.

We show now that if $A, B \in 2^{\Omega_p[\psi]}$ with $A \cap B = \emptyset$, then $\theta_\psi(a)(\Gamma)(A) + \theta_\psi(a)(\Gamma)(B) = \theta_\psi(a)(\Gamma)(A \cup B)$. Let $A = [\![\phi_1]\!]$, $B = [\![\phi_2]\!]$ with $\phi_1, \phi_2 \in \mathcal{L}[\psi]$ and $\vdash \phi_1 \to \neg\phi_2$. Let $x_1 = \theta_\psi(a)(\Gamma)(A)$, $x_2 = \theta_\psi(a)(\Gamma)(B)$ and $x = \theta_\psi(a)(\Gamma)(A \cup B)$. We prove that $x_1 + x_2 = x$.

Suppose that $x_1 + x_2 < x$. Then, there exist $\epsilon_1, \epsilon_2 \in \mathbb{Q}^+$ such that $x'_1 + x'_2 < x$, where $x'_i = x_i + \epsilon_i$ for $i = 1, 2$. From the definition of x_i, $\neg L^a_{x'_i}\phi_i \in \Gamma^\infty$. Further, using (A4), we obtain $\neg L^a_{x'_1+x'_2}(\phi_1 \vee \phi_2) \in \Gamma^\infty$, implying that $x'_1 + x'_2 \geq x$ - contradiction.

Suppose that $x_1 + x_2 > x$. Then, there exist $\epsilon_1, \epsilon_2 \in \mathbb{Q}^+$ such that $x''_1 + x''_2 > x$, where $x''_i = x_i - \epsilon_i$ for $i = 1, 2$. But the definition of x_i implies that $L^a_{x''_i}\phi_i \in \Gamma^\infty$. Further, (A3) gives $L^a_{x''_1+x''_2}(\phi_1 \vee \phi_2) \in \Gamma^\infty$, i.e. $x''_1 + x''_2 \leq x$ - contradiction.

◀

Now we can prove the Truth Lemma.

▶ **Lemma 9** (Truth Lemma). *If $\phi \in \mathcal{L}[\psi]$, then $[\mathcal{M}_\psi, \Gamma \Vdash \phi$ iff $\phi \in \Gamma]$.*

Proof. Induction on the structure of ϕ. The only nontrivial case is $\phi = L^a_r\phi'$.
($\Longrightarrow$) Suppose that $\mathcal{M}_\psi, \Gamma \Vdash \phi$ and $\phi \notin \Gamma$. Hence $\neg\phi \in \Gamma$. Let $y = min\{r \in \mathbb{Q}_p : \neg L^a_r\phi \in \Gamma\}$. Then, from $\neg L^a_r\phi' \in \Gamma$, we obtain $r \geq y$. But $\mathcal{M}_\psi, \Gamma \Vdash L^a_r\phi'$ is equivalent with $\theta_\psi(a)(\Gamma)([\![\phi']\!]) \geq r$, i.e. $a^\Gamma_{\phi'} \geq r$. On the other hand, from Lemma 6, $a^\Gamma_{\phi'} < y$ - contradiction.
($\Longleftarrow$) If $L^a_r\phi' \in \Gamma$, then $r \leq a^\Gamma_\phi$ and $r \leq \theta_\psi(a)(\Gamma)([\![\phi]\!])$. Hence, $\mathcal{M}_\psi, \Gamma \Vdash L^a_r\phi$. ◀

The previous lemma implies the small model property for our logic.

▶ **Theorem 10** (Small model property). *For any $\mathcal{L}(\mathcal{A})$-consistent formula ϕ, there exists $\mathcal{M} \in \mathfrak{M}$ with finite support of cardinality bound by the structure of ϕ, and there exists $m \in supp(\mathcal{M})$ such that $\mathcal{M}, m \Vdash \phi$.*

The small model property proves the (weak) completeness of the axiomatic system.

▶ **Theorem 11** (Weak Completeness). *The axiomatic system of $\mathcal{L}(\mathcal{A})$ is complete with respect to the Markovian semantics, i.e. if $\Vdash \psi$, then $\vdash \psi$.*

Proof. We have that [$\Vdash \psi$ implies $\vdash \psi$] is equivalent with [$\nvdash \psi$ implies $\nVdash \psi$], that is equivalent with [the consistency of $\neg\psi$ implies the existence of a model $(\mathcal{M}, m)$ for $\neg\psi$] and this is guaranteed by the finite model property. ◀

(B1):	$\vdash L^a_0\phi$
(B2):	$\vdash L^a_{r+s}\phi \to \neg M^a_r\phi,\ s>0$
(B3):	$\vdash \neg L^a_r\phi \to M^a_r\phi$
(B4):	$\vdash \neg L^a_r(\phi\wedge\psi)\wedge\neg L^a_s(\phi\wedge\neg\psi)\to\neg L^a_{r+s}\phi$
(B5):	$\vdash \neg M^a_r(\phi\wedge\psi)\wedge\neg M^a_s(\phi\wedge\neg\psi)\to\neg M^a_{r+s}\phi$
(S1):	If $\vdash\phi\to\psi$ then $\vdash L^a_r\phi\to L^a_r\psi$
(S2):	If $\forall r<s, \vdash\phi\to L^a_r\psi$ then $\vdash\phi\to L^a_s\psi$
(S3):	If $\forall r>s, \vdash\phi\to M^a_r\psi$ then $\vdash\phi\to M^a_s\psi$
(S4):	If $\forall r>s, \vdash\phi\to L^a_r\psi$ then $\vdash\phi\to\bot$

Table 2 The axiomatic system of $\mathcal{L}^+(\mathcal{A})$

5.2 Axiomatization for $\mathcal{L}^+(\mathcal{A})$

Table 2 contains a Hilbert-style axiomatization for $\mathcal{L}^+(\mathcal{A})$.

Notice the differences between these axioms and the axioms in Table 1. First of all Axiom (A2) had to be enforced by (B2) and (B3) which depict the connection between the two stochastic operators. In the probabilistic case these relations are encoded by the duality rule $M^a_r\phi = L^a_{1-r}\neg\phi$ and by the axiom $\vdash L^a_r\phi\to\neg L^a_s\neg\phi$ for $r+s<1$; these two are not sound for stochastic models. Rule (A3) has been itself enforced by (B5). We also have an extra Archimedean rule for M^a_r. We prove below that all the theorems of $\mathcal{L}(A)$ are also theorems of $\mathcal{L}^+(\mathcal{A})$ and we state some theorems of $\mathcal{L}^+(\mathcal{A})$ that are central for the weak completeness proof of $\mathcal{L}^+(\mathcal{A})$.

▶ **Lemma 12.** *1.* $\vdash L^a_{r+s}\phi\to L^a_r\phi$, *2.* $\vdash M^a_r\phi\to M^a_{r+s}\phi$,
3. $\vdash L^a_r(\phi\wedge\psi)\wedge L^a_s(\phi\wedge\neg\psi)\to L^a_{r+s}\phi$, *4.* $\vdash M^a_r(\phi\wedge\psi)\wedge M^a_s(\phi\wedge\neg\psi)\to M^a_{r+s}\phi$,
5. $\vdash\neg M^a_r\phi\to L^a_r\phi$, *6.* $\vdash M^a_r\phi\to\neg L^a_{r+s}\phi,\ s>0$,
7. If $\vdash\phi\to\psi$, *then* $\vdash M^a_r\psi\to M^a_r\phi$.

▶ **Theorem 13** (Soundness). *The axiomatic system of* $\mathcal{L}^+(\mathcal{A})$ *is sound for the Markovian semantics, i.e., for any* $\phi\in\mathcal{L}^+(\mathcal{A})$, *if* $\vdash\phi$ *then* $\Vdash\phi$.

The finite model property for $\mathcal{L}^+(\mathcal{A})$ is proved, similarly to the case of $\mathcal{L}(\mathcal{A})$, by using the filtration method. In what follows we will not reproduce the entire construction already presented for $\mathcal{L}(\mathcal{A})$, but we only emphasize the major differences.

We keep the notations introduced before with the only differences that for an arbitrary $\phi\in\mathcal{L}^+(\mathcal{A})$, the definition of the modal depth of ϕ also includes $md(M^a_r\psi)=md(\psi)+1$ and $gr(\phi)$, $max(\phi)$ and $act(\phi)$ take into account, in addition, the indexes of the operators of type M^a_r that appear in ϕ. With these modifications, we define $\mathcal{L}^+_n(A)$, for any integer n and $A\subseteq\mathcal{A}$, as before and for $\Lambda\subseteq\mathcal{L}^+(\mathcal{A})$, $[\Lambda]_n=\Lambda\cup\{\phi\in\mathcal{L}^+_n(\mathcal{A}):\Lambda\vdash\phi\}$.

Consider a consistent formula $\psi\in\mathcal{L}^+(\mathcal{A})$ with $gr(\psi)=n$ and $act(\psi)=A$. We define $\mathcal{L}^+[\psi]=\{\phi\in\mathcal{L}^+_n(A)\mid max(\phi)\le max(\psi), md(\phi)\le md(\psi)\}$. Let $\Omega[\psi]$ be the set of $\mathcal{L}^+[\psi]$-maximally consistent sets of formulas. We remake the construction done in the previous subsection for $\mathcal{L}(\mathcal{A})$.

The first important difference with respect to the previous case appears due to (B2): for each $\Lambda\in\Omega[\psi]$, $\phi\in\Lambda$ and $a\in\mathcal{A}$, there exist $s,t\in\mathbb{Q}_p$, $s<t$, such that $L^a_s\phi, M^a_r\phi\in\Lambda^+$. Secondly, for any $\Gamma\in\Omega_p[\psi]$, $\phi\in\mathcal{L}^+[\psi]$ and $a\in A$, there exist $x=max\{r\in\mathbb{Q}_p:\neg M^a_r\phi\in\Gamma\}$, $y=min\{r\in\mathbb{Q}_p:M^a_r\phi\in\Gamma\}$ and $y=x+1/p$. In effect, in the correspondent of Lemma 7, one can prove that there exists $z=sup\{r\in\mathbb{Q}:L^a_r\phi\in\Gamma^\infty\}=inf\{r\in\mathbb{Q}:\neg L^a_r\phi\in\Gamma^\infty\}=inf\{r\in\mathbb{Q}:M^a_r\phi\in\Gamma^\infty\}=sup\{r\in\mathbb{Q}:\neg M^a_r\phi\in\Gamma^\infty\}$.

As before, we denote z by a^{Γ}_{ϕ} and we proceed with the definition of the model $\mathcal{M}_{\psi}$.

▶ **Lemma 14.** *If $\theta_{\psi} : \mathcal{A} \to [\Omega_p[\psi] \to \Delta(\Omega_p[\psi], 2^{\Omega_p[\psi]})]$ is defined for arbitrary $a \in \mathcal{A}$, $\Gamma \in \Omega_q[\psi]$ and $\phi \in \mathcal{L}^+[\psi]$ by $\theta_{\psi}(a)(\Gamma)([\![\phi]\!]) = a^{\Gamma}_{\phi}$, then $\mathcal{M}_{\psi} = (\Omega_p[\psi], 2^{\Omega_p[\psi]}, \theta_{\psi}) \in \mathfrak{M}$.*

This last result allows us to prove the Truth Lemma for $\mathcal{L}^+(\mathcal{A})$.

▶ **Lemma 15** (Truth Lemma). *If $\phi \in \mathcal{L}^+[\psi]$, then $[\mathcal{M}_{\psi}, \Gamma \Vdash \phi$ iff $\phi \in \Gamma]$.*

The proof of Lemma 15 requires, in addition to the proof of Lemma 9, the case $\phi = M^a_r \phi'$ which is proved similarly to the case $\phi = L^a_r \phi'$.

As before, the truth lemma implies the finite model property and the weak completeness theorem for $\mathcal{L}^+(\mathcal{A})$ with Markovian semantics.

▶ **Theorem 16** (Small model property). *For any $\mathcal{L}^+(\mathcal{A})$-consistent formula ϕ, there exists $\mathcal{M} \in \mathfrak{M}$ with finite support of cardinality bound by the structure of ϕ, and there exists $m \in supp(\mathcal{M})$ such that $\mathcal{M}, m \Vdash \phi$.*

▶ **Theorem 17** (Weak Completeness). *The axiomatic system of $\mathcal{L}^+(\mathcal{A})$ is complete with respect to the Markovian semantics, i.e. if $\Vdash \psi$, then $\vdash \psi$.*

6 From bisimulation to the metric space of logical formulas

To start with, we state that the logical equivalences induced by $\mathcal{L}(\mathcal{A})$ and by $\mathcal{L}^+(\mathcal{A})$ on the class of CMPs coincide with stochastic bisimulation. The proofs follow closely the proof of the corresponding result for probabilistic systems [8, 10, 20] and consist in showing that the negation free-fragment of $\mathcal{L}(\mathcal{A})$ characterizes stochastic bisimulation while the negation and M^a_r do not differentiate bisimilar processes.

▶ **Theorem 18** (Logical characterization of stochastic bisimulation). *Let $\mathcal{M} = (M, \Sigma, \tau), \mathcal{M}' = (M', \Sigma', \tau') \in \mathfrak{M}$, $m \in M$ and $m' \in M'$. The following assertions are equivalent.*
1. *$(\mathcal{M}, m) \sim (\mathcal{M}', m')$;*
2. *For any $\phi \in \mathcal{L}(\mathcal{A})$, $\mathcal{M}, m \Vdash \phi$ iff $\mathcal{M}', m' \Vdash \phi$;*
3. *For any $\phi \in \mathcal{L}^+(\mathcal{A})$, $\mathcal{M}, m \Vdash \phi$ iff $\mathcal{M}', m' \Vdash \phi$.*

One of the main motivation for studying quantitative logics for probabilistic and stochastic processes was, since the first papers on this subject [17, 16], the characterization of stochastic/-probabilistic bisimulation. In the context of Theorem 18, one can turn the bisimulation question into a series of model-checking problems. But the concept of stochastic/probabilistic bisimulation is a very strict concept: it only verifies whether two processes have identical behaviours. In applications we need instead to know whether two processes that may differ by a small amount in the real-valued parameters (rates or probabilities) have similar behaviours. To solve this problem a class of pseudometrics have been proposed in the literature [6, 20], to measure how similar two processes are in terms of stochastic/probabilistic behaviour.

Because these pseudometrics are quantitative extensions of bisimulation, they can be defined relying on the quantitative logics. Thus, for a class $\mathfrak{P}$ of stochastic or probabilistic processes and for a quantitative logic $\mathcal{L}$ that characterizes the bisimulation of processes, the pseudometric can be induced by a function $d : \mathfrak{P} \times \mathcal{L} \to [0, 1]$ which extends the (characteristic function of the) satisfiability relation $\Vdash$: $\mathfrak{P} \times \mathcal{L} \to \{0, 1\}$; the function d evaluates the *"degree of satisfiability"* [6, 20].

In this paper we work with the function $d : \mathfrak{P} \times \mathcal{L} \to [0, 1]$, defined below for the set $\mathfrak{P}$ of CMPs and $\mathcal{L} = \mathcal{L}^+(\mathcal{A})$ (or $\mathcal{L} = \mathcal{L}(\mathcal{A})$).

$d((\mathcal{M},m),\top) = 0$,
$d((\mathcal{M},m),\neg\phi) = 1 - d((\mathcal{M},m),\phi)$,
$d((\mathcal{M},m),\phi\wedge\psi) = max\{d((\mathcal{M},m),\phi), d((\mathcal{M},m),\psi)\}$,
$d((\mathcal{M},m),L^a_r\phi) = \langle r, \theta(a)(m)([\![\phi]\!])\rangle$,
$d((\mathcal{M},m),M^a_r\phi) = \langle \theta(a)(m)([\![\phi]\!]), r\rangle$,
where for arbitrary $a,b\in\mathbb{R}_+$, $\langle a,b\rangle = (a-b)/a$ if $a(a-b)>0$ and $\langle a,b\rangle = 0$ otherwise.
The results presented in this section relay on the fact that d, as most of the functions that quantify satisfiability for stochastic or probabilistic logics, is defined on top of the transition function θ. For this reason, these results can be similarly proved for other bisimulation pseudometrics.

The first result states that d characterizes stochastic bisimulation.

▶ **Lemma 19.** *If* $(\mathcal{M},m),(\mathcal{M}',m')\in\mathfrak{P}$, *then*

$$(\mathcal{M},m)\sim(\mathcal{M}',m') \text{ iff } [\forall\phi\in\mathcal{L}, d((\mathcal{M},m),\phi) = d((\mathcal{M}',m'),\phi)].$$

Proof. ($\Longrightarrow$) Induction on ϕ. The Boolean cases are trivial and the cases $\phi = L^a_r\psi$ and $\phi = M^a_r\psi$ derive from the fact that $\theta(a)(m)([\![\psi]\!]) = \theta'(a)(m')([\![\psi]\!])$.
($\Longleftarrow$) For an arbitrary $\phi\in\mathcal{L}$, $\forall r\in\mathbb{Q}$, $d((\mathcal{M},m),L^a_r\phi) = d((\mathcal{M}',m'),L^a_r\phi)$; and for r big enough $d((\mathcal{M},m),L^a_r\phi) = 1-\theta(a)(m)([\![\phi]\!])/r$, $d((\mathcal{M}',m'),L^a_r\phi) = 1-\theta'(a)(m')([\![\phi]\!])/r$. Hence, $\theta(a)(m)([\![\phi]\!]) = \theta'(a)(m')([\![\phi]\!])$ which implies $(\mathcal{M},m)\sim(\mathcal{M}',m')$. ◀

As we have anticipated, a function $d:\mathfrak{P}\times\mathcal{L}\to[0,1]$ which characterizes bisimulation in the sense of Lemma 19, induces a distance between stochastic processes, $D:\mathfrak{P}\times\mathfrak{P}\to[0,1]$ by

$$D(P,P') = sup\{|d(P,\phi) - d(P',\phi)|, \phi\in\mathcal{L}\}, \text{ for arbitrary } P,P'\in\mathfrak{P}.$$

D is, indeed, a pseudometric and its kernel is the stochastic bisimulation.

▶ **Lemma 20.** $D:\mathfrak{P}\times\mathfrak{P}\to[0,1]$ *defined before is a pseudometric such that*

$$D(P,P') = 0 \text{ iff } P\sim P'.$$

Similarly, one can use d to define a pseudometric $\overline{d}:\mathcal{L}\times\mathcal{L}\to[0,1]$ over the space of logical formulas by

$$\overline{d}(\phi,\psi) = sup\{|d(P,\phi) - d(P,\psi)|, P\in\mathfrak{P}\}, \text{ for arbitrary } \phi,\psi\in\mathcal{L}.$$

▶ **Lemma 21.** $\overline{d}:\mathcal{L}\times\mathcal{L}\to[0,1]$ *defined before is a pseudometric and*

$$\overline{d}(\phi,\psi) = \overline{d}(\neg\phi,\neg\psi).$$

Proof. We prove that it satisfies the triangle inequality. We have
$sup\{|d((\overline{\Omega},\Gamma),\phi) - d((\overline{\Omega},\Gamma),\psi)|\} + sup\{|d((\overline{\Omega},\Gamma),\psi) - d((\overline{\Omega},\Gamma),\rho)|\} \geq sup\{|d((\overline{\Omega},\Gamma),\phi) - d((\overline{\Omega},\Gamma),\psi)| + |d((\overline{\Omega},\Gamma),\psi) - d((\overline{\Omega},\Gamma),\rho)|\} \geq sup\{|d((\overline{\Omega},\Gamma),\phi) - d((\overline{\Omega},\Gamma),\rho)|\}$. ◀

This construction allow us to introduce the first robustness theorem.

▶ **Theorem 22** (Strong Robustness). *For arbitrary* $\phi,\psi\in\mathcal{L}$ *and* $P\in\mathfrak{P}$,

$$d(P,\psi)\leq d(P,\phi) + \overline{d}(\phi,\psi).$$

Proof. From the definition of $\overline{d}$ we have that $d(P,\psi) - d(P,\phi)\leq\overline{d}(\phi,\psi)$. ◀

Similar constructions can be done for any class of stochastic or probabilistic models for which it has been defined a correspondent logic that characterizes bisimulation. But in spite of the obvious utility of the robustness theorem, in most of the cases such a result is not computable due to the definition of $\overline{d}$ that involves the quantification over the entire class of continuous Markov processes.

This is exactly where the sound and complete axiomatizations of $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}^+(\mathcal{A})$ for the Markovian semantics and the finite model properties play their role. In what follows, we use the construction of the small model for an $\mathcal{L}$-consistent formula presented in the previous section[4] to effectively compute an approximation of $\overline{d}$ within a given error $\varepsilon > 0$. Below we reuse the notations of section 5.

Let Ω be the set of the $\mathcal{L}$-maximally consistent sets of formulas. For arbitrary $\Gamma^\infty \in \Omega$, $a \in \mathcal{A}$ and $\phi \in \mathcal{L}$, let

$$a_\phi^{\Gamma^\infty} = sup\{r \in \mathbb{Q} : L_r^a \phi \in \Gamma^\infty\} = inf\{r \in \mathbb{Q} : \neg L_r^a \phi \in \Gamma^\infty\} =$$

$$inf\{r \in \mathbb{Q} : M_r^a \phi \in \Gamma^\infty\} = sup\{r \in \mathbb{Q} : \neg M_r^a \phi \in \Gamma^\infty\}.$$

The existence of these inf and sup and their equalities can be proved as in Lemma 6 (4).

▶ **Lemma 23** (Extended Truth Lemma). *If $\theta : \mathcal{A} \to [\Omega \to \Delta(\Omega, 2^\Omega)]$ is defined for arbitrary $a \in \mathcal{A}$, $\Gamma^\infty \in \Omega$ and $\phi \in \mathcal{L}$ by $\theta(a)(\Gamma^\infty)([\![\phi]\!]) = a_\phi^{\Gamma^\infty}$, then $\mathcal{M}_\mathcal{L} = (\Omega, 2^\Omega, \theta) \in \mathfrak{M}$. Moreover, for arbitrary $\phi \in \mathcal{L}$,*

$$\mathcal{M}_\mathcal{L}, \Gamma^\infty \Vdash \phi \text{ iff } \phi \in \Gamma^\infty.$$

The proof of this lemma is the sum of the proofs of the lemmas 8, 9, 14 and 15.

The next lemma states that $\overline{d}$ can be characterized by only using the processes of $\mathcal{M}_\mathcal{L}$. In this way it relates $\overline{d}$ to provability, as these processes are $\mathcal{L}$-maximally consistent sets of formulas.

▶ **Lemma 24.** *For arbitrary $\phi, \psi \in \mathcal{L}$,*

$$\overline{d}(\phi, \psi) = sup\{|d((\mathcal{M}_\mathcal{L}, \Gamma^\infty), \phi) - d((\mathcal{M}_\mathcal{L}, \Gamma^\infty), \psi)|, \Gamma^\infty \in \Omega\}.$$

Proof. Any $(\mathcal{M}, m) \in \mathfrak{M}$ satisfies a maximally-consistent set of formulas, hence there exists $\Gamma^\infty \in \Omega$ such that $(\mathcal{M}, m) \sim (\mathcal{M}_\mathcal{L}, \Gamma^\infty)$, i.e., for any $\phi \in \mathcal{L}$, $d((\mathcal{M}, m), \phi) = d((\mathcal{M}_\mathcal{L}, \Gamma^\infty), \phi)$. ◀

In what follows we reduce the quantification space to the domain of a finite model. For an arbitrary consistent formula $\psi \in \mathcal{L}$, let $\mathcal{M}_\psi = (\Omega_p[\psi], 2^{\Omega_p[\psi]}, \theta_\psi) \in \mathfrak{M}$ be the model of ψ constructed in the previous section; we call p the *parameter of* $\mathcal{M}_\psi$.
Let $\widetilde{d} : \mathcal{L} \times \mathcal{L} \to [0, 1]$ be defined as follows.
$\widetilde{d}(\phi, \psi) = max\{|d((\mathcal{M}_{\phi\wedge\psi}, \Gamma), \phi) - d((\mathcal{M}_{\phi\wedge\psi}, \Gamma), \psi)|, \Gamma \in \Omega_p[\phi \wedge \psi]\}$ if $\phi \wedge \psi$ is consistent,
$\widetilde{d}(\phi, \psi) = max\{|d((\mathcal{M}_{\neg(\phi\wedge\psi)}, \Gamma), \phi) - d((\mathcal{M}_{\neg(\phi\wedge\psi)}, \Gamma), \psi)|, \Gamma \in \Omega_p[\neg(\phi \wedge \psi)]\}$, otherwise.

▶ **Lemma 25.** *For arbitrary $\phi, \psi \in \mathcal{L}$,*

$$\overline{d}(\phi, \psi) \leq \widetilde{d}(\phi, \psi) + 2/p.$$

[4] These results hold for both $\mathcal{L} = \mathcal{L}(\mathcal{A})$ and $\mathcal{L} = \mathcal{L}^+(\mathcal{A})$.

Proof. To prove the inequality, we return to the notations of lemmas 6 and 7. We have $x, y \in \mathbb{Q}_p$, $y = x + 1/p$ and $x \leq z < y$. This implies that for any $\phi \in \mathcal{L}[\psi]$, $|d((\mathcal{M}_{\mathcal{L}}, \Gamma^\infty), \phi) - d((\mathcal{M}_{\phi\wedge\psi}, \Gamma), \phi)| \leq 1/p$. Consequently, for arbitrary $\phi, \psi \in \mathcal{L}$, $|d((\mathcal{M}_{\mathcal{L}}, \Gamma^\infty), \phi) - d((\mathcal{M}_{\mathcal{L}}, \Gamma^\infty), \psi)| \leq |d((\mathcal{M}_{\phi\wedge\psi}, \Gamma), \phi) - d((\mathcal{M}_{\phi\wedge\psi}, \Gamma), \psi)| + 2/p$, which proofs our inequality (if $\phi \wedge \psi$ is inconsistent we take $\mathcal{M}_{\neg(\phi\wedge\psi)}$). ◀

This last result finally allows us to prove a weaker version of the robustness theorem which evaluates $d((\mathcal{M}, m), \psi)$ from $d((\mathcal{M}, m), \phi)$, based on $\widetilde{d}(\phi, \psi)$ and a given error.

▶ **Theorem 26** (Weak Robustness). *For arbitrary $\phi, \psi \in \mathcal{L}$ and $P \in \mathfrak{P}$,*

$$d(P, \psi) \leq d(P, \phi) + \widetilde{d}(\phi, \psi) + 2/p,$$

where p is the parameter of $\mathcal{M}_{\phi\wedge\psi}$ if $\phi \wedge \psi$ is consistent, or of $\mathcal{M}_{\neg(\phi\wedge\psi)}$ otherwise.

Because $\mathcal{M}_{\phi\wedge\psi}$ (or $\mathcal{M}_{\neg(\phi\wedge\psi)}$) is finite, $\widetilde{d}(\phi, \psi)$ can be computed and the error $2/p$ can be controlled while constructing $\mathcal{M}_{\phi\wedge\psi}$. Hence, we can evaluate $d(P, \phi)$ from $d(P, \psi)$. This is useful when P is infinite or very large and it is expensive to repeatedly evaluate $d(P, \phi)$ for various ϕ. Instead, our theorem allows us to evaluate $d(P, \psi)$ from $d(P, \phi)$ that we can get, for instance, using statistical model checking techniques.

7 Conclusions and future works

We have introduced Continuous Markovian Logic, a multimodal logic designed to specify quantitative and qualitative properties of continuous Markov processes. CML is endowed with operators that approximate the rates of the labelled transitions of CMPs. This logic characterizes the stochastic bisimulation of CMPs.

We have presented two sound and complete Hilbert-style axiomatizations: one for the entire CML and one for its fragment without M_r^a-operators. These axiomatic systems are significantly different from the probabilistic case and from each other. The two completeness proofs relay on the finite model properties. The constructions of the finite models adapts the filtration method of modal logics to stochastic settings, where a series of specific problems had to be solved. The small model constructions and the complete axiomatizations allow us to approach syntactically the problems of bisimulation-distances, which before in the literature have only been treated semantically. In effect, we can define a distance between logical formulas that allows us to prove the robustness theorems.

This paper opens a series of interesting research questions regarding the relation between satisfiability, provability and metric semantics summarized in [18]. There are many open questions related to the definition of $\overline{d}$ and to the structure of the metric space of formulas. One of the problems, that we postpone for future work, is finding a classification of the functions d to reflect properties of $\overline{d}$. For instance, we have a partial result showing that if d satisfies some continuity conditions, then $\overline{d}$ characterizes the logical equivalence between positive formulas (formulas without negation), i.e., $[\overline{d}(\phi, \psi) = 0$ iff $\Vdash \phi \leftrightarrow \psi]$ [18]. There exist, however, distances enjoying even stronger properties such as $[\Vdash \phi \to \psi$ iff $\forall P \in \mathfrak{P}$, $d(P, \psi) \leq d(P, \phi)]$. Each of these metrics organizes the set of logical formulas as a metric space with specific topological properties. The complete axiomatization is probably the key for classifying these structures and for understanding the relationship between the topological space of models and the topological space of logical formulas.

References

1 R. Aumann. *Interactive epistemology: Probability.* Int. J. Game Theory, 28:301-314, 1999.
2 R. Blute, J. Desharnais, A. Edalat, P. Panangaden. *Bisimulation for labeled Markov processes.* In Proc. of LICS'97, IEEE Press, 1997.
3 L. Cardelli, R. Mardare. *The Measurable Space of Stochastic Processes.* In Proc. QEST2010, IEEE Computer Society, pp. 171-180. 2010
4 L. Cardelli, K. G. Larsen, R. Mardare. *Modular Markovian Logic.* In Proc. of ICALP 2011, LNCS 6756, 2011, to appear.
5 V. Danos, J. Desharnais, F. Laviolette, P. Panangaden. *Bisimulation and Coconguence for Probabilistic Systems.* Inf. and Comp. 204(4):503-523, 2006.
6 J. Desharnais, Labelled Markov Processes. PhD thesis, McGill University, 1999.
7 E. Doberkat, *Stochastic Relations. Foundations for Markov Transition Systems.* Chapman and Hall/CRC, 2007.
8 J. Desharnais, A. Edalat, P. Panangaden. *A logical characterization of bisimulation for labeled Markov processes.* In Proc of LICS'98, IEEE Press, 1998.
9 J. Desharnais, A. Edalat, P. Panangaden. *Bisimulation for labeled Markov Processes.* Inf. and Comp., 179(2):163-193, 2002.
10 J. Desharnais, P. Panangaden. *Continous Stochastic Logic Characterizes Bisimulation of Continuous-time Markov Processes.* JLAP. 56:99-115, 2003.
11 R. Fagin, J. Halpern. *Reasoning about knowledge and probability.* J. ACM, 41:340-367, 1994.
12 J. Harsanyi. *Games with incomplete information played by bayesian players, part one.* Management Sci., 14:159-182, 1967.
13 A. Heifetz, P. Mongin. *Probability Logic for Type Spaces.* Games and Economic Behavior 35, 3153, 2001.
14 C. Kupke, D. Pattinson. *On Modal Logics of Linear Inequalities.* In Proc of AiML 2010.
15 K. Sen, M. Viswanathan, G. Agha. *Statistical Model Checking of Black-Box Probabilistic Systems.* LNCS 3114:202-215, 2004.
16 K.G. Larsen and A. Skou. *Compositional Verification of Probabilistic Processes.* LNCS 630:456-471, 1992.
17 K.G. Larsen and A. Skou. *Bisimulation through probabilistic testing.* Inf. and Comp., 94:1-28, 1991.
18 R. Mardare. *Markovian Logics: The Metric Space of Logical Formulas.* The Bulletin of Symbolic Logic, the meeting report of Logic Colloquium 2011, to appear.
19 L.S. Moss, I.D. Viglizzo. *Harsanyi type spaces and final coalgebras constructed from satisfied theories.* ENTCS 106:279-295, 2004.
20 P. Panangaden. *Labelled Markov Processes.* Imperial College Press, 2009.
21 F. van Breugel, F. Worrell. *A behavioural pseudometric for probabilistic transition systems.* TCS 331:115-142, 2005.
22 H. L. S. Younes. *Ymer: A Statistical Model Checker.* LNCS 3576:429-433, 2005.
23 C. Zhou., *A complete deductive system for probability logic with application to Harsanyi type spaces.* PhD thesis, Indiana University, 2007.

The Focused Calculus of Structures

Kaustuv Chaudhuri, Nicolas Guenot, and Lutz Straßburger

INRIA & LIX/École Polytechnique
Route de Saclay, 91128 Palaiseau, France
{kaustuv,nguenot,lutz}@lix.polytechnique.fr

Abstract

The *focusing* theorem identifies a complete class of sequent proofs that have no inessential non-deterministic choices and restrict the essential choices to a particular normal form. Focused proofs are therefore well suited both for the search and for the representation of sequent proofs. The *calculus of structures* is a proof formalism that allows rules to be applied deep inside a formula. Through this freedom it can be used to give analytic proof systems for a wider variety of logics than the sequent calculus, but standard presentations of this calculus are too permissive, allowing too many proofs. In order to make it more amenable to proof search, we transplant the focusing theorem from the sequent calculus to the calculus of structures. The key technical contribution is an incremental treatment of focusing that avoids trivializing the calculus of structures. We give a direct inductive proof of the completeness of the focused calculus of structures with respect to a more standard unfocused form. We also show that any focused sequent proof can be represented in the focused calculus of structures, and, conversely, any proof in the focused calculus of structures corresponds to a focused sequent proof.

1998 ACM Subject Classification F.4.1 Proof theory

Keywords and phrases Focusing, polarity, calculus of structures, linear logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.159

1 Introduction

Logic has traditionally been seen as a means of representing and systematizing mathematical knowledge, but it is increasingly being used to encode and reason about formal systems—programming languages, process calculi, transition systems, *etc.*—that are inherently computational. In this use of logic, the syntax of proofs is important to build correspondences between the proofs in logic and the computations of the encoded systems, also known as the problem of *representational adequacy*. An adequate encoding is not only manifestly correct, *i.e.*, it represents all and only the computations of the encoded system, but is also useful as a device to automate the reasoning in and about the encoded system. In standard proof systems such as Gentzen's *sequent calculus*, it is usually impossible to construct adequate encodings: there are more proofs than computational traces, because the inference rules are more non-deterministic than the computational steps.

In recent years the *focusing* theorem of Andreoli [1] has been used to create certain "normal forms" of sequent proofs where the question of representational adequacy becomes considerably easier, often trivial, for focused proofs. Focusing was originally developed for (classical) linear logic but has since been extended to a wide spectrum of logics [3, 14]. The essential observation of focusing is that sequent rules have certain natural permutative affinities that can be exploited to fuse logical connectives into larger *synthetic connectives*; for example, the synthetic connective $- \otimes (- \oplus -)$ behaves as a ternary connective instead of as a composition of two binary connectives. The problem of representational adequacy

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 159–173

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

is reduced to that of encoding the computational steps in such a way that they correspond exactly to the synthetic connectives.

This technique has been successful for the standard classical, intuitionistic, and linear logics where the sequent calculus is most natural. The sequent calculus is, however, inherently limited in its expressivity: it cannot be used to give *analytic* (*i.e.*, cut-free) proof systems for many modal and non-commutative logics that have been used for program safety, operational semantics, or linguistics. The common feature of many of these logics is that they rely on *deep inference* or the ability to perform deduction inside a formula. Proof systems for many such logics need to generalize the sequent calculus; some popular generalizations include: hypersequents [2], nested sequents [5], or the display calculus [4]. The most permissive, and therefore most expressive, of such generalizations is the *calculus of structures* [10, 11, 6] that does not differentiate between formulas and sequents and can therefore perform deduction anywhere inside a formula. Besides the increased expressivity, proofs in the calculus of structures can also exploit features that are not available in the other shallower formalisms; for example, they can be exponentially smaller than sequent proofs [7], or they can be decomposed in a number of ways [20].

A main distinguishing feature of the calculus of structures is that it divides the sequent rules into smaller components, thereby introducing more non-deterministic choices. The sequent calculus operates on entire (multi-)sets of formulas, with a single sequent rule able to split whole contexts multiplicatively or test for the absence of certain elements. The rules of the calculus of structures, on the other hand, perform such operations incrementally on fragments of contexts. Some of the inessential choices introduced by this incremental nature can be removed by restricting the *syntactic congruence* in the original formulation of the calculus [10, 11], leading to the system LS for classical propositional linear logic (outlined in Section 3.1), which can be seen as a variant of the original formulation in [17, 18] that is more amenable to automation. One important feature of LS is that contraction is the sole rule that makes proofs unbounded, and it permutes below all other rules (Proposition 8), a property that is crucial for the cut-elimination result for LS (also presented in Section 3.1). The contraction-free fragment of LS is therefore decidable.

Yet, despite its more parsimonious design, LS is still at least as non-deterministic as the unfocused sequent calculus. It is natural to ask if a result similar to focusing can tame LS in the same way that the sequent system LLK (without cut) for linear logic (Figure 1) was tamed to produce the focused system LLKF (Figure 2). For the purely multiplicative fragment, this question has already been investigated in [9], but the strategy there seems difficult to generalize. In this paper (in Section 3.2) we construct a focused variant of the calculus LS, called LSF, for full classical propositional linear logic. It uses the technical device of polarized formulas [12]; polarities make the synthetic connectives manifest in the syntax, and the rules of LSF are organized to respect polarity, *i.e.*, to never introduce a polarity change that did not already exist. Synthetic connectives are thus preserved in LSF proofs.

To show LSF complete with respect to LS in its own right, *i.e.*, that any LS proof can be turned into an LSF proof, we build a equivalent synthetic variant of LSF called LSS (see Section 3.2). A special rule that breaks the polarity restriction is added to LSS to represent unfocused LS proofs directly, and then this rule is shown to be admissible in LSS. We thus have a simple *internal* proof of completeness of LSS (and hence of LSF) with respect to LS. This style of showing completeness of focusing for the calculus of structures can pave the way for focused variants of other logics that lack an analytic sequent system. Although we limit our attention to classical propositional linear logic in this paper, we consider it an important future work to extend our focusing result to logics for which focusing in terms of

$$\mathsf{id}\ \frac{}{\vdash a, \overline{a}} \qquad \mathsf{ct}\ \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A} \qquad \mathsf{wk}\ \frac{\vdash \Gamma}{\vdash \Gamma, ?A} \qquad \mathsf{cut}\ \frac{\vdash \Gamma, A \quad \vdash \Delta, (A)^\perp}{\vdash \Gamma, \Delta}$$

$$\otimes\ \frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B} \qquad 1\ \frac{}{\vdash 1} \qquad \oplus_1\ \frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B} \qquad \oplus_2\ \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B} \qquad !\ \frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}$$

$$\parr\ \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B} \qquad \bot\ \frac{\vdash \Gamma}{\vdash \Gamma, \bot} \qquad \&\ \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B} \qquad \top\ \frac{}{\vdash \Gamma, \top} \qquad ?\ \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}$$

Figure 1 LLK: a one-sided single-zoned sequent calculus for classical propositional linear logic

the sequent calculus is inapplicable. This includes logics like BV [10], the logic of bunched implications [16], and various modal logics.

We also compare LSF and LLKF by first showing that any LLKF proof can be simulated in LSF (in Section 4.1), *i.e.*, that LSF is powerful enough to represent focused sequent proofs. Then we also give an algorithm to *extract* an LLKF proof from any LSF proof that is unique up to permutations between negative rules (in Section 4.2). These two results justify the use of the adjective "focused" for LSF. Together with the completeness of LSF for LS, this result can be used to give an alternative proof of completeness of LLKF for LLK.

2 The Sequent Calculus and Focusing for Linear Logic

We begin with a quick overview of the standard sequent calculus and the focusing theorem for classical propositional linear logic whose formulas $(A, B, \ldots)$ have the following grammar:

$$A, B \ ::= \ a \mid A \otimes B \mid 1 \mid A \oplus B \mid 0 \mid !A \mid \overline{a} \mid A \parr B \mid \bot \mid A \,\&\, B \mid \top \mid ?A$$

The *atoms* $(a, b, \ldots)$ are drawn from some countably infinite set. Formulas are in negation normal form, with the negation of a written as $\overline{a}$, and negation of formulas $(-)^\perp$ as follows:

$$(a)^\perp = \overline{a} \quad (A \otimes B)^\perp = (A)^\perp \parr (B)^\perp \quad (1)^\perp = \bot \quad (A \oplus B)^\perp = (A)^\perp \,\&\, (B)^\perp \quad (0)^\perp = \top \quad (!A)^\perp = ?\,(A)^\perp$$
$$(\overline{a})^\perp = a \quad (A \parr B)^\perp = (A)^\perp \otimes (B)^\perp \quad (\bot)^\perp = 1 \quad (A \,\&\, B)^\perp = (A)^\perp \oplus (B)^\perp \quad (\top)^\perp = 0 \quad (?A)^\perp = !\,(A)^\perp$$

The standard sequent calculus for linear logic, called LLK and shown in Figure 1, is given in terms of one-sided single-zoned *sequents* of the form $\vdash \Gamma$ where Γ is a *context* (a multi-set of formulas).

▶ **Theorem 1** (cut elimination). *The* cut *rule is admissible in* LLK $\setminus$ {cut}. ◀

Let us now sketch the focused variant of LLK, called LLKF. For this, the formulas are divided into two *polarity* classes—*positive* and *negative*—based on the permutation properties of their sequent rules. The negative formulas have invertible rules, *i.e.*, rules that may be applied whenever the formula occurs in the context, while the rules for the positive formulas are sensitive to the order of application of rules and are therefore generally non-invertible. Following [12], we syntactically distinguish these two classes and mediate between them by a pair of *shift* connectives (↓ and ↑):

$$\begin{array}{rcll} P, Q & ::= & a \mid P \otimes Q \mid 1 \mid P \oplus Q \mid 0 \mid !N \mid \downarrow N & \text{(Positive Formulas)} \\ N, M & ::= & \overline{a} \mid N \parr M \mid \bot \mid N \,\&\, M \mid \top \mid ?P \mid \uparrow P & \text{(Negative Formulas)} \end{array}$$

Here, $(\downarrow N)^\perp = \uparrow (N)^\perp$ and $(\uparrow P)^\perp = \downarrow (P)^\perp$. We use the following contexts for LLKF:

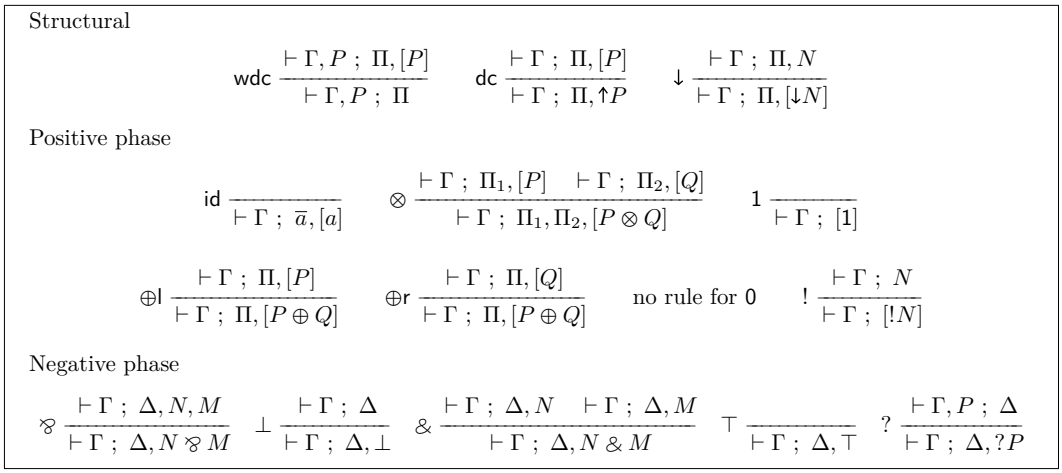


Figure 2 LLKF: a two-zoned focused variant of cut-free LLK

$$\begin{array}{lll} \Gamma & ::= & \cdot \mid \Gamma, P & \text{(Positive Sequent Contexts)} \\ \Delta & ::= & \cdot \mid \Delta, N & \text{(Negative Sequent Contexts)} \\ \Pi & ::= & \cdot \mid \Pi, \uparrow P \mid \Pi, \overline{a} & \text{(Reactive Sequent Contexts)} \end{array}$$

LLKF proofs consist of alternating maximal *phases* based on the polarity of the principal formulas. These two phases are represented by two different sequent forms, given below. We follow Andreoli's original two-zoned (*dyadic*) convention for presenting the system because it is the most common style in presenting focused proof systems.

$$\begin{array}{ll} \vdash \Gamma \;;\; \Delta & \text{(Negative Sequents)} \\ \vdash \Gamma \;;\; \Pi, [P] & \text{(Positive Sequents)} \end{array}$$

The sequent $\vdash \Gamma \;;\; \cdot, [P]$ is abbreviated as $\vdash \Gamma \;;\; [P]$.

The focused rules of inference are shown in Figure 2. The most important rules are the *decision* rules wdc and dc that begin[1] a positive phase; in this phase, the focused formula (written inside [ ]) is principal, and the focus persists on the principal operands if they are of the same polarity. All essential choices are confined to this phase; they include: disjunctive choice (for $\oplus$), multiplicative choice (for $\otimes$), possible failures (for atoms, 1, and !, if the context is not of the correct form), and guaranteed failure (for 0, which has no rules). The positive phase switches to the negative phase with the rules for $\downarrow$ or !. Observe that in the negative phase the rules can be applied in any order, and none of the negative rules can fail to apply. When no more negative rules can apply, a decision rule must be applied to restart the cycle. There are no structural rules of weakening or contraction because the rules treat the *unrestricted context* Γ as a set; in particular, contraction is part of the wdc rule.

The soundness of LLKF with respect to cut-free LLK is straightforward: forgetting the polarities, the focusing distinctions in sequents, and the rules $\{\mathsf{dc}, \downarrow\}$; prefixing the elements of Γ with ? and using ct and wk to account for its additive treatment in the $\otimes$, 1 and id rules; and replacing wdc with the sequence ct then ? on the focused formula produces valid LLK proofs from LLKF proofs.

▶ **Notation 2.** Write $\lfloor P \rfloor$ (resp. $\lfloor N \rfloor$) for the unpolarized formula obtained from P (resp. N) by erasing all occurrences of $\downarrow$ and $\uparrow$. Similarly we define $\lfloor \Delta \rfloor$.

[1] As usual, the intended reading of sequent rules is from conclusion to premises.

▶ **Theorem 3** (completeness of focusing). *If* $\vdash \lfloor \Delta \rfloor$ *in* LLK, *then* $\vdash \cdot \; ; \; \Delta$ *in* LLKF.

There are many ways to prove this theorem; we refer the interested reader to one of the standard approaches [13, 15]. One interesting feature of all such proofs is their unusual complexity forced by the rigidity of the focusing calculus. It is easier to show the completeness of focusing in the sequent calculus with more synthetic approaches [8].

3 Linear Logic in the Calculus of Structures

The calculus of structures is based on the observation that the connectives of linear logic preserve logical entailment, and therefore, any valid implication in the logic can be turned into a rewrite step on any subformula. Hence, there is no need to maintain a distinction between the connectives used in formulas, and structural meta-connectives such as the comma used to write sequents, or the meta conjunction among the premises of a binary rule. The original formulations of the calculus of structures [10, 11, 6] used *structures*, which are formulas modulo a syntactic congruence. We deviate from this tradition and use just the formulas, *i.e.*, we remove the syntactic congruence. Then, inference rules are allowed to operate on any subformula. These rules are therefore written in terms of *formula contexts* $(\xi, \zeta, \ldots)$, which are formulas with a single hole (written $\{\,\}$), *i.e.*, they have the following grammar:

$$\xi, \zeta \;::=\; \{\,\} \mid A \star \xi \mid \xi \star A \mid\, !\xi \mid\, ?\xi \qquad \text{(Formula Contexts)}$$

where $\star$ can stand for any binary connective ($\otimes$, $\oplus$, $\parr$, or $\&$). We write $\xi\{A\}$ for the formula formed by replacing the single occurrence of $\{\,\}$ in ξ with the formula A. For example, if ξ is $!\,(a \,\&\, (\{\,\} \otimes b)) \parr ?c$ and A is $\overline{a} \oplus b$, then $\xi\{A\}$ is $!\,(a \,\&\, ((\overline{a} \oplus b) \otimes b)) \parr ?c$.

A *derivation* $\mathscr{D}$ in a system S with premise A and conclusion B is a rewriting path from A to B, using the rules in S. It is usually depicted as $\begin{array}{c}\mathsf{s}\\ \| \\ B\end{array}\!\!\begin{array}{c}A\\ \mathscr{D}\end{array}$. A *proof* $\mathscr{P}$ in a system S, depicted as $\begin{array}{c}\mathsf{s}\\ \| \\ B\end{array}\!\!\mathscr{P}$, is a derivation in S with premise 1.

3.1 The Unfocused Systems SLS and LS

The inference rules of the system SLS are given in Figure 3. The first two columns constitute the *multiplicative fragment*, the next two columns the *exponential fragment*, and the last two columns the *additive fragment*. The multiplicative and exponential fragment constitute system SELS, which is a variant of the system studied in [19]. The first four rows in Figure 3 constitute the *down fragment* of SLS, denoted by SLS↓, and the last four rows the *up fragment*, denoted by SLS↑. The down fragment corresponds to the cut-free version of the system, and following the tradition, we will call it LS. Each rule in either fragment is the dual of some rule in the other fragment, where the duals of a rule are formed by exchanging the premise and conclusion and negating both. Note that the two rules sl and sr (read *switch left* and *switch right*) are self-dual and therefore part of both fragments.[2] The rules $\mathsf{ai}_{\downarrow}$ and $\mathsf{ai}_{\uparrow}$, called *atomic identity* and *atomic cut*, have the following general versions:

$$\mathsf{i}_{\downarrow}\,\frac{\xi\{1\}}{\xi\{A \parr (A)^{\perp}\}} \qquad \mathsf{i}_{\uparrow}\,\frac{\xi\{(A)^{\perp} \otimes A\}}{\xi\{\perp\}}$$

[2] Note that our system SLS is slightly different from the presentation in [17, 18]. The reason for the differences is that we get stronger results (e.g., the down fragment does not need associativity for $\otimes$, $\oplus$, and $\&$), their proofs become simpler, and the relation to the focused systems is more evident.

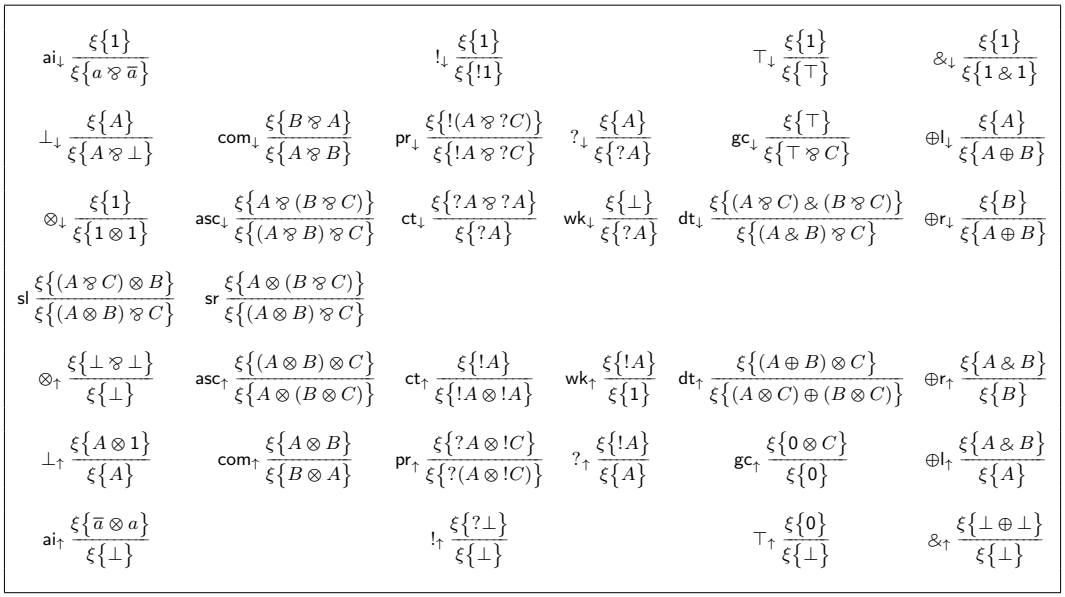


Figure 3 SLS, a symmetric calculus of structures for classical propositional linear logic. The fragment containing the first four rows is called LS.

Like in the sequent calculus, the general identity rule is derivable. By duality the same is true for the general cut rule. We have the following proposition, which is standard for systems in the calculus of structures (see. e.g., [17]).

▶ **Proposition 4.** *The rule* $\mathsf{i}_{\downarrow}$ *is derivable in* SLS$\downarrow$*, and the rule* $\mathsf{i}_{\uparrow}$ *is derivable in* SLS$\uparrow$*. Furthermore, every rule in* SLS$\uparrow$ *is derivable in* SLS$\downarrow + \mathsf{i}_{\uparrow}$*, and dually, every rule in* SLS$\downarrow$ *is derivable in* SLS$\uparrow + \mathsf{i}_{\downarrow}$*.* ◀

By an easy induction on the size of the proofs, one can show the following implications, expressing the relation to the sequent calculus.

▶ **Proposition 5.** *A formula* A *is provable in* LLK *with cut if and only if it is provable in* SLS*. And if* A *is provable in* LLK *without cut, then it is provable in* LS*.* ◀

We can now use Theorem 1 to show that provability in SLS implies provability in LS and that provability in LS, implies provability in LLK without cut.

▶ **Theorem 6** (cut elimination)**.** *If a formula* A *is provable in* SLS *then it is provable in* LS*.*

▶ **Corollary 7.** *The rule* $\mathsf{i}_{\uparrow}$ *is admissible for* LS*.* ◀

The proof given in [17] for Theorem 6 relies on the sequent calculus and Theorem 1. In the following, we present a proof that is internal to SLS, *i.e.*, not using the sequent calculus. Due to lack of space, we can only give a sketch—all details can be found in [18]. First, observe that in any derivation $\mathscr{D}$ in LS, all the instances of the contraction rule can be permuted to the bottom. This can be shown by an easy inductive argument.

▶ **Proposition 8.** *For every* $\mathsf{LS}\left\|\begin{smallmatrix}A\\B\end{smallmatrix}\right.\mathscr{D}$ *there is a* B' *such that* $\mathsf{LS}\setminus\{\mathsf{ct}_{\downarrow}\}\left\|\begin{smallmatrix}A\\B'\end{smallmatrix}\right.\mathscr{D}'$ *and* $\{\mathsf{ct}_{\downarrow}\}\left\|\begin{smallmatrix}B'\\B\end{smallmatrix}\right.\mathscr{D}''$*.* ◀

For the internal cut-elimination proof of SLS, we will use a technique called *splitting*, first used in [10]. The central ingredients are Lemmas 10 – 12 and Lemma 14 below. Lemmas

10 – 12 say how the connectives behave in a *shallow* context, and Lemma 14 says how a general *deep* context can be reduced to a shallow one. For formally stating these lemmas, we need the notions of *linear killing context* and *killing context*, denoted by $\lambda\langle\ \rangle$ and $\kappa\langle\ \rangle$, respectively, and generated by this grammar:

$$\begin{array}{lll} \lambda & ::= & \top \mid \{\ \} \mid \lambda \mathbin{\&} \lambda \mid \lambda \otimes 1 \mid 1 \otimes \lambda \qquad \text{(Linear Killing Contexts)} \\ \kappa & ::= & \top \mid \{\ \} \mid \kappa \mathbin{\&} \kappa \mid \kappa \otimes 1 \mid 1 \otimes \kappa \mid\ !\kappa \qquad \text{(Killing Contexts)} \end{array}$$

We write $\lambda\langle\ \rangle^n$ (resp. $\kappa\langle\ \rangle^n$) to indicate that there are exactly n occurrences of $\{\ \}$. Then, we write $\lambda\langle A_1, \ldots, A_n\rangle$ (resp. $\kappa\langle A_1, \ldots, A_n\rangle$) for the formula obtained from $\lambda\langle\ \rangle^n$ (resp. $\kappa\langle\ \rangle^n$) by replacing, from left to right, the n occurrences of $\{\ \}$ by the formulas $A_1, \ldots, A_n$. The two main properties of killing contexts are summarized in the following lemma.

▶ **Lemma 9.** *Let $A, B_1, \ldots, B_n$, and $\lambda\langle\ \rangle^n$ and $\kappa\langle\ \rangle^n$ be given.*

1. *If $B_1, \ldots, B_n$ are provable in* LS*, then so are $\lambda\langle B_1, \ldots, B_n\rangle$ and $\kappa\langle B_1, \ldots, B_n\rangle$.*
2. *There are derivations* $\dfrac{\lambda\langle A \parr B_1, \ldots, A \parr B_n\rangle}{A \parr \lambda\langle B_1, \ldots, B_n\rangle}\,\mathsf{LS}\!\parallel$ *and* $\dfrac{\kappa\langle ?A \parr B_1, \ldots, ?A \parr B_n\rangle}{?A \parr \kappa\langle B_1, \ldots, B_n\rangle}\,\mathsf{LS}\!\parallel$. ◀

We can now state the splitting lemmas.

▶ **Lemma 10** (binary splitting)**.** *Let A, B, and K be formulas.*

1. *If $(A \mathbin{\&} B) \parr K$ is provable in* LS*, then so are $A \parr K$ and $B \parr K$.*
2. *If $(A \oplus B) \parr K$ is provable in* LS*, then there is an $n \geq 0$ and $K_1, \ldots, K_n$ and $\lambda\langle\ \rangle^n$, such that* $\dfrac{\lambda\langle K_1, \ldots, K_n\rangle}{K}\,\mathsf{LS}\!\parallel$ *and for all $i \leq n$ we have* $\dfrac{}{A \parr K_i}\,\mathsf{LS}\!\parallel$ *or* $\dfrac{}{B \parr K_i}\,\mathsf{LS}\!\parallel$.
3. *If $(A \otimes B) \parr K$ is provable in* LS*, then there are $n \geq 0$ and $K_{A1}, K_{B1}, \ldots, K_{An}, K_{Bn}$ and $\lambda\langle\ \rangle^n$, such that* $\dfrac{\lambda\langle K_{A1} \parr K_{B1}, \ldots, K_{An} \parr K_{Bn}\rangle}{K}\,\mathsf{LS}\!\parallel$ *and* $\dfrac{}{A \parr K_{Ai}}\,\mathsf{LS}\!\parallel$ *and* $\dfrac{}{B \parr K_{Bi}}\,\mathsf{LS}\!\parallel$ *for all $i \leq n$.*

▶ **Lemma 11** (unit and atomic splitting)**.** *Let x be an atom or a negated atom, and let K be a formula.*

4. *If $1 \parr K$ is provable in* LS*, then there is a $\lambda\langle\ \rangle^n$ and a derivation* $\dfrac{\lambda\langle \bot, \ldots, \bot\rangle}{K}\,\mathsf{LS}\!\parallel$.
5. *If $\bot \parr K$ is provable in* LS*, then so is K.*
6. *If $0 \parr K$ is provable in* LS*, then there is a $\lambda\langle\ \rangle^n$ and a derivation from $\lambda\langle \top, \ldots, \top\rangle$ to K in* LS*.*
7. *If $x \parr K$ is provable in* LS*, then there is a $\lambda\langle\ \rangle^n$ and a derivation* $\dfrac{\lambda\langle x^\bot, \ldots, x^\bot\rangle}{K}\,\mathsf{LS}\!\parallel$.

▶ **Lemma 12** (exponential splitting)**.** *Let A and K be formulas.*

8. *If $!A \parr K$ is provable in* LS*, then there are $n \geq 0$ and $K_1, \ldots K_n$ and $\lambda\langle\ \rangle^n$, such that* $\dfrac{\lambda\langle K_1, \ldots, K_n\rangle}{K}\,\mathsf{LS}\!\parallel$ *and for all $i \leq n$ we have* $\dfrac{}{A \parr K_i}\,\mathsf{LS}\!\parallel$ *with $K_i = \bot$ or $K_i = ?K_{i1} \parr \cdots \parr ?K_{ih_i}$ for some $h_i \geq 1$.*
9. *If $?A \parr K$ is provable in $\mathsf{LS} \setminus \{\mathsf{ct}_\downarrow\}$, then either K is provable in $\mathsf{LS} \setminus \{\mathsf{ct}_\downarrow\}$, or there are $n \geq 1$ and $K_1, \ldots K_n$ and $\kappa\langle\ \rangle^n$, such that* $\dfrac{\kappa\langle K_1, \ldots, K_n\rangle}{K}\,\mathsf{LS}\setminus\{\mathsf{ct}_\downarrow\}\!\parallel$ *and* $\dfrac{}{A \parr K_i}\,\mathsf{LS}\setminus\{\mathsf{ct}_\downarrow\}\!\parallel$ *for all $i \leq n$.*

All three splitting lemmas are proved in a similar way by an induction on the size of the given proof, and a case analysis on the bottommost rule instance. Although the statements of the splitting lemmas are different from the ones in [18], the proofs are almost literally the same. The purpose of the splitting lemmas is to prove the following lemma, which says that the rules of the up-fragment are admissible in a shallow context.

▶ **Lemma 13.** *Let K be a formula and let* $\mathsf{r}_{\uparrow}\dfrac{\xi\{F\}}{\xi\{G\}}$ *be a rule in* SLS↑*. If $F \parr K$ is provable in* LS*, then so is $G \parr K$.*

This is proved by using splitting to decompose the proof of $F \parr K$ into smaller pieces which can then be rearranged to build a proof of $G \parr K$. For the rules $\mathsf{pr}_{\uparrow}$ and $!_{\uparrow}$, we also need Proposition 8.

For proving Theorem 6, we need to extend Lemma 13 to general contexts. This is done by the *context reduction* lemma, whose proof is a straightforward induction on the context, repeatedly applying splitting.

▶ **Lemma 14** (context reduction)**.** *Let A be a formula and ξ be a context in which $\{\ \}$ does not appear inside the scope of a ?-modality. If $\xi\{A\}$ is provable in* LS*, then there exist an $n \geq 0$, a killing context $\kappa\langle\ \rangle$, and formulas $K_{A1}, \ldots, K_{An}$, such that*

$$\kappa\langle C \parr K_{A1}, \ldots, C \parr K_{An}\rangle \quad \text{for every formula } C \quad \text{and} \quad A \parr K_{Ai} \quad \text{for every } i \leq n.$$

$$\begin{array}{c} \mathsf{LS}\| \\ \xi\{C\} \end{array} \qquad \begin{array}{c} \mathsf{LS}\| \\ A \parr K_{Ai} \end{array}$$

◀

▶ **Lemma 15.** *Let* $\mathsf{r}_{\uparrow}\dfrac{\xi\{F\}}{\xi\{G\}}$ *be a rule in* SLS↑ *and let ξ be a context in which $\{\ \}$ does not appear inside the scope of a ?-modality. If $\xi\{F\}$ is provable in* LS*, then so is $\xi\{G\}$.*

This follows immediately from Lemma 13 and Lemma 14. In order to deal with ?-contexts, we use the following lemma, proved by a simple rule permutation argument: any rule applied inside the scope of a ? can be permuted up until it leaves the scope of the ?-modality.

▶ **Lemma 16** (?-reduction)**.** *For every proof $\mathscr{P}$ in* SLS *there is a proof $\mathscr{P}'$ in* SLS *with the same conclusion as $\mathscr{P}$, such that in $\mathscr{P}'$ no inference rule is applied inside the scope of a ?-modality.* ◀

Now Theorem 6 can be shown by first applying Lemma 16 and then eliminating all up-rules, starting with the topmost one, using Lemma 15.

3.2 LSF and LSS: Polarized, Focused, and Synthetic Variants of LS

In this section we study two complete polarized and focused variants of LS. Like in Andreoli's original formulation of focusing in the sequent calculus [1], we keep the general form of the rules of LS but modify them to respect polarity. The resulting calculus, called LSF, can be seen to be related to LS in the same way that LLKF is related to LLK. (In Section 4 below, we formalize the comparison between LLKF and LSF.) Just as in the sequent calculus, the proofs of completeness of the focusing restriction will become more manageable in a synthetic formulation of LSF, that we call LSS, which we present immediately after LSF.

Because LSF uses polarized formulas, the contexts in LSF are sensitive to the polarity of their holes. We use π and ρ for *positive formula contexts*, *i.e.*, $\pi\{P\}$ is a well-formed polarized formula for any positive formula P. Likewise, we use ν and μ for negative formula contexts. Note that the polarity of $\pi\{P\}$ (resp. $\nu\{N\}$) need not itself be positive (resp. negative).

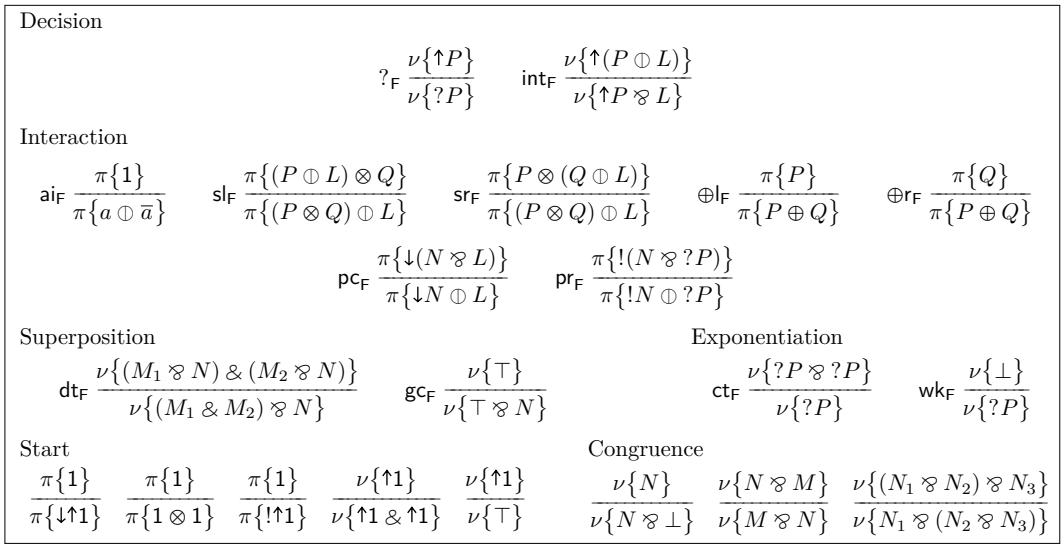


Figure 4 LSF: a polarized and focused variant of LS

Inside positive contexts, we will use a notational device to mark the foci. To motivate this notation, consider the properties of the foci in LLKF sequents: they *interact* with the context by splitting it (for $\otimes$), by testing it for emptiness (for 1 and $!$), or by checking for the presence of negated atoms (for id). For LSF, these interactions need to be made incremental—formula by formula—because the formulas of the corresponding sequent context may not (yet) be present in a $\parr$ relation at the point of the switch rules.

▶ **Definition 17.** An *interaction formula* (or simply an *interaction*) is a positive formula of the form $P \oplus L$. We call P in $P \oplus L$ the *focus* of the interaction, and L its *spine*.

Here, we use L to stand for *reactive formulas*, which correspond to the formulas that occur in LLKF sequents $\vdash \Gamma\ ;\ \Pi$, which are precisely those sequents that are introduced by the decision rules wdc and dc. Recall that such sequents represent formulas of the form $\parr(?\Gamma, \Pi)$. Reactive formulas, and their duals, the *active formulas*, therefore have this grammar:

$$\begin{array}{lll} R & ::= & a \mid\ !N \mid \downarrow N \qquad \text{(Active Formulas)} \\ L & ::= & \overline{a} \mid\ ?P \mid \uparrow P \qquad \text{(Reactive Formulas)} \end{array}$$

Figure 4 lists the inference rules of LSF. A proof in LSF is a derivation with premise 1 or $\uparrow 1$. The start rules in Figure 4 define what it means to finish an LSF proof. The first start rule removes a pair of shifts from a 1, and the other four are polarized versions of the rules $\otimes_{\downarrow}$, $!_{\downarrow}$, $\&_{\downarrow}$, and $\top_{\downarrow}$ of LS (Figure 3). Interactions are created by the $\mathsf{int}_{\mathsf{F}}$ rule, which corresponds to dc in LLKF. When the focus of the interaction involves a polarity shift, the interaction dissipates into an ordinary $\parr$ using pc_{F}, which is the analogue of the $\downarrow$ rule of LLKF. In order to remain true to the spirit of LS, we keep contraction and decision as separate rules instead of building a specialized version of $\mathsf{int}_{\mathsf{F}}$ that incorporates contraction. This lets us preserve the permutability of contraction (Proposition 8) even in the focused setting. To retain completeness, the $?_{\mathsf{F}}$ rule derelicts a $?$ to a $\uparrow$. The remaining rules for interactions follow the shape of the focus of the interaction, just as in the sequent calculus. For example, for $\otimes$, the rules sl_{F} and sr_{F} (that are the focused versions of the sl and sr rules of LS) send the spine of the interaction into one of the components of the focus. The remaining (non-interaction) rules are simply the direct polarity-respecting translations of the LS rules.

▶ **Theorem 18** (soundness). *For any N, if N is provable in* LSF*, then $\lfloor N \rfloor$ is provable in* LS*.*

Proof. Just replace $P \mathbin{⦶} L$ with $P \parr L$ and erase the polarity shifts. ◀

To show completeness, we will now move to a synthetic variant of LSF, called LSS, that keeps a sequence of interactions on subformulas of a focus together. While the correspondence with LLKF is clearer in LSF without this synthetic step, the proof of completeness is drastically simplified with synthetic formulations, a phenomenon that has also been observed for focusing in the sequent calculus [8].

The key observation needed to produce a synthetic variant of LSF is the following: in an interaction formula $P \mathbin{⦶} L$, the spine L is switched (using $\mathsf{sl_F}$ and $\mathsf{sr_F}$) *deep* inside P until the focus of the interaction become active. During this switching, any $\oplus$-formulas in the focus are destructed by removing (using $\oplus\mathsf{l_F}$ and $\oplus\mathsf{r_F}$) one of its operands. Thus, we can define a special *tensor context*, written using $\pi_\otimes$ and $\rho_\otimes$, with this grammar:

$$\pi_\otimes \quad ::= \quad \{\,\} \mid \pi_\otimes \otimes P \mid P \otimes \pi_\otimes \qquad \text{(Tensor Contexts)}$$

Note that, because $\pi_\otimes$ contains no shift or exponential connectives, any substitution $\pi_\otimes\{P\}$ is itself positive. Tensor contexts allow us to write the following synthetic forms of the interaction rules:

$$\mathsf{sai_F}\,\frac{\nu\{\uparrow\pi_\otimes\{1\}\}}{\nu\{\uparrow\pi_\otimes\{a\} \parr \overline{a}\}} \qquad \mathsf{spc_F}\,\frac{\nu\{\uparrow\pi_\otimes\{\downarrow(N \parr L)\}\}}{\nu\{\uparrow\pi_\otimes\{\downarrow N\} \parr L\}} \qquad \mathsf{spr_F}\,\frac{\nu\{\uparrow\pi_\otimes\{!(N \parr ?P)\}\}}{\nu\{\uparrow\pi_\otimes\{!N\} \parr ?P\}}$$

▶ **Definition 19.** The system LSS is $\mathsf{LSF} \setminus \{\mathsf{int_F}, \mathsf{ai_F}, \mathsf{sl_F}, \mathsf{sr_F}, \mathsf{pc_F}, \mathsf{pr_F}\} \cup \{\mathsf{sai_F}, \mathsf{spc_F}, \mathsf{spr_F}\}$.

▶ **Theorem 20.** *Any formula is provable in* LSS *if and only if it is provable in* LSF*.*

Proof. Each instance of $\mathsf{sai_F}$, $\mathsf{spc_F}$, or $\mathsf{spr_F}$ can be derived by one of

$$\mathsf{ai_F}\,\frac{\nu\{\uparrow\pi_\otimes\{1\}\}}{\nu\{\uparrow\pi_\otimes\{a \mathbin{⦶} \overline{a}\}\}} \qquad \mathsf{pc_F}\,\frac{\nu\{\uparrow\pi_\otimes\{\downarrow(M \parr L)\}\}}{\nu\{\uparrow\pi_\otimes\{\downarrow M \mathbin{⦶} L\}\}} \qquad \mathsf{pr_F}\,\frac{\nu\{\uparrow\pi_\otimes\{!(M \parr ?P)\}\}}{\nu\{\uparrow\pi_\otimes\{!M \mathbin{⦶} ?P\}\}}$$

$$\mathsf{int_F}\,\frac{\nu\{\uparrow(\pi_\otimes\{a\} \mathbin{⦶} \overline{a})\}}{\nu\{\uparrow\pi_\otimes\{a\} \parr \overline{a}\}} \qquad \mathsf{int_F}\,\frac{\nu\{\uparrow(\pi_\otimes\{\downarrow M\} \mathbin{⦶} L)\}}{\nu\{\uparrow\pi_\otimes\{\downarrow M\} \parr L\}} \qquad \mathsf{int_F}\,\frac{\nu\{\uparrow(\pi_\otimes\{!M\} \mathbin{⦶} ?P)\}}{\nu\{\uparrow\pi_\otimes\{!M\} \parr ?P\}}$$

(each $\mathsf{int_F}$ with $\{\mathsf{sl_F}, \mathsf{sr_F}\} \Vdash \mathscr{D}$)

where $\mathscr{D}$ can be constructed by a straightforward induction on $\pi_\otimes$.

In the other direction, note that $⦶$ is not commutative, *i.e.*, the order of the spines is fixed in iterations of $⦶$. We can therefore permute any LSF proof to guarantee that the focus of any interaction isn't itself an interaction. Finally, we permute all instances of $\oplus\mathsf{l_F}$ and $\oplus\mathsf{r_F}$ as low as possible in the LSF proof so that all interactions are introduced by $\mathsf{ai_F}$, $\mathsf{pc_F}$ or $\mathsf{pr_F}$. The synthetic rules $\mathsf{sai_F}$, $\mathsf{spc_F}$ and $\mathsf{spr_F}$ can now be easily recovered. ◀

For giving the proof of completeness of LSS with respect to LS, we proceed by inductive transformation of LS proofs in three steps. First, we rewrite the instances of the switch rules in LS to respect the restriction of the spines to reactive formulas, which corresponds to applying negative rules eagerly like in LLKF. Second, we use an auxiliary rule, called $\mathsf{ps_F}$, that breaks the polarity restrictions by means of an extra pair of shifts in the premise. This rule allows us to transform any LS proof into a proof in $\mathsf{LSS} \cup \{\mathsf{ps_F}\}$. And third, we show that $\mathsf{ps_F}$ can be eliminated from $\mathsf{LSS} \cup \{\mathsf{ps_F}\}$.

For the first step, let LS^r stand for LS where the rules sl and sr (see Figure 3) are replaced by the following rules sl^r and sr^r, respectively:

$$\mathsf{sl}^\mathsf{r}\,\frac{\xi\{(A \parr \lfloor L \rfloor) \otimes B\}}{\xi\{(A \otimes B) \parr \lfloor L \rfloor\}} \qquad \mathsf{sr}^\mathsf{r}\,\frac{\xi\{A \otimes (B \parr \lfloor L \rfloor)\}}{\xi\{(A \otimes B) \parr \lfloor L \rfloor\}}$$

Recall that L stands for reactive formulas. The following can be shown by an easy induction:

▶ **Lemma 21.** *The rules*

$$\mathsf{cc}_{\downarrow}\,\frac{\xi\{(A \,\&\, B) \otimes C\}}{\xi\{(A \otimes C) \,\&\, (B \otimes C)\}} \qquad \mathsf{ga}_{\downarrow}\,\frac{\xi\{\top \otimes A\}}{\xi\{\top\}} \qquad \bar{\perp}_{\downarrow}\,\frac{\xi\{A \parr \perp\}}{\xi\{A\}} \qquad \bar{\mathsf{dt}}_{\downarrow}\,\frac{\xi\{(A \,\&\, B) \parr C\}}{\xi\{A \parr C\} \,\&\, (B \parr C)} \qquad \bar{\mathsf{gc}}_{\downarrow}\,\frac{\xi\{\top \parr C\}}{\xi\{\top\}}$$

are admissible in LS^{r}. ◀

▶ **Lemma 22.** *If a formula A is provable in* LS*, then it is also provable in* LS^{r}.

Proof. Let the *size* of an instance of sl or sr with conclusion $\xi\{(A \otimes B) \parr C\}$ be defined as the number of symbols used in C. For transforming an LS proof into an LS^{r} proof we take two steps:

$$\mathsf{LS}\left\|\,\mathscr{D}_1\right. \atop A \qquad\longrightarrow\qquad \mathsf{LS}^{\mathsf{r}\cup\{\mathsf{cc}_{\downarrow},\mathsf{ga}_{\downarrow},\bar{\perp}_{\downarrow},\bar{\mathsf{dt}}_{\downarrow},\bar{\mathsf{gc}}_{\downarrow}\}}\left\|\,\mathscr{D}_2\right. \atop A \qquad\xrightarrow{\text{Lemma 21}}\qquad \mathsf{LS}^{\mathsf{r}}\left\|\,\mathscr{D}_3\right. \atop A$$

For the first step, proceed by induction on the multi-set of the sizes of all switch instances in $\mathscr{D}_1$, under multi-set ordering, showing that all instances of sl and sr can be reduced to sl^{r} and sr^{r}. Any instance of sl but not of sl^{r} can be replaced by one of the following derivations, reducing the size. (Note that we omitted some instances of $\mathsf{asc}_{\downarrow}$ and $\mathsf{com}_{\downarrow}$.)

$$\mathsf{sl}\,\frac{\mathsf{sl}\,\dfrac{(B \parr (G \parr H)) \otimes C}{((B \parr G) \otimes C) \parr H}}{(B \otimes C) \parr (G \parr H)} \qquad \bar{\perp}_{\downarrow}\,\frac{(B \parr \perp) \otimes C}{B \otimes C}\;\perp_{\downarrow}\,\frac{}{(B \otimes C) \parr \perp}$$

$$\mathsf{dt}_{\downarrow}\,\frac{\mathsf{sl}\,\dfrac{\mathsf{cc}_{\downarrow}\,\dfrac{\bar{\mathsf{dt}}_{\downarrow}\,\dfrac{(B \parr (G \,\&\, H)) \otimes C}{((B \parr G) \,\&\, (B \parr H)) \otimes C}}{((B \parr G) \otimes C) \,\&\, ((B \parr H) \otimes C)}}{((B \parr G) \otimes C) \,\&\, ((B \otimes C) \parr H)}}{((B \otimes C) \parr G) \,\&\, ((B \otimes C) \parr H)}}{(B \otimes C) \parr (G \,\&\, H)} \qquad \bar{\mathsf{gc}}_{\downarrow}\,\frac{(B \parr \top) \otimes C}{\mathsf{gc}_{\downarrow}\,\dfrac{\mathsf{ga}_{\downarrow}\,\dfrac{\top \otimes C}{\top}}{(B \otimes C) \parr \top}}$$

The cases for sr are similar. ◀

The second step of the transformation of LS proofs to LSS proofs involves the following *partial switch* synthetic rule:

$$\mathsf{ps}_{\mathsf{F}}\,\frac{\nu\{\uparrow\pi_{\otimes}\{\downarrow(\uparrow P \parr L)\}\}}{\nu\{\uparrow\pi_{\otimes}\{P\} \parr L\}}$$

This rule has more shifts in the premise than in the conclusion and is therefore not derivable in LSS. It can permute above any rule in $\mathsf{LSS} \setminus \{\mathsf{sai}_{\mathsf{F}}, \mathsf{spc}_{\mathsf{F}}, \mathsf{spr}_{\mathsf{F}}\}$. Before we can show that ps_{F} can always be eliminated from any proof of LSS, we need a lemma stating that $\downarrow\uparrow$ can be removed from an LSS proof with the use of ps_{F}.

▶ **Lemma 23.** *If there is a proof $\mathscr{D}$ of a negative formula $\pi\{\downarrow\uparrow P\}$ in $\mathsf{LSS}\cup\{\mathsf{ps}_{\mathsf{F}}\}$, then there is a proof of $\pi\{P\}$ in $\mathsf{LSS}\cup\{\mathsf{ps}_{\mathsf{F}}\}$ of at most the same height as $\mathscr{D}$.*

Proof. Proceed by induction on the height of $\mathscr{D}$. In the base case, $\pi\{\downarrow\uparrow P\}$ is $\uparrow\downarrow\uparrow 1$ and the result is immediate. In the general case, consider the bottommost rule instance r in $\mathscr{D}$; in most cases the induction hypothesis is directly applicable so the pair of shifts can simply be removed. The only interesting case is where r is a matching instance of $\mathsf{spc}_{\mathsf{F}}$. We replace it by an instance of ps_{F} as follows:

$$\mathsf{spc}_{\mathsf{F}}\,\frac{\nu\{\uparrow\pi_{\otimes}\{\downarrow(\uparrow P \parr L)\}\}}{\nu\{\uparrow\pi_{\otimes}\{\downarrow\uparrow P\} \parr L\}} \qquad\longrightarrow\qquad \mathsf{ps}_{\mathsf{F}}\,\frac{\nu\{\uparrow\pi_{\otimes}\{\downarrow(\uparrow P \parr L)\}\}}{\nu\{\uparrow\pi_{\otimes}\{P\} \parr L\}}$$ ◀

▶ Remark. Note that we also have the converse: if there is a proof of $\pi\{P\}$ in $\mathsf{LSS}\cup\{\mathsf{ps}_{\mathsf{F}}\}$, then there is also a proof of $\pi\{\downarrow\uparrow P\}$ in $\mathsf{LSS}\cup\{\mathsf{ps}_{\mathsf{F}}\}$.

▶ **Lemma 24.** *For every N, if $\lfloor N \rfloor$ is provable in LS^{r}, then N is provable in $\mathsf{LSS} \cup \{\mathsf{ps_F}\}$.*

Proof. Proceed by induction on the height of the LS^{r} proof $\mathscr{D}$ of $\lfloor N \rfloor$ to build a proof of N in $\mathsf{LSS} \cup \{\mathsf{ps_F}\}$. The base case, where $\lfloor N \rfloor$ is 1, is trivial. Now make a case analysis for the bottommost rule instance r in $\mathscr{D}$. In most cases, we can simply replace r with the corresponding rule in LSS, and appeal to the induction hypothesis on the proof above r. The four interesting cases involve r being an instance of $\mathsf{ai}_{\downarrow}$, sl^{r}, sr^{r}, or $\mathsf{pr}_{\downarrow}$. We can apply the induction hypothesis to the proof above r and glue the result to one the following rule instances depending on the case:

$$\mathsf{sai_F}\,\frac{\nu\{\uparrow 1\}}{\nu\{\uparrow a \parr \overline{a}\}} \qquad \mathsf{ps_F}\,\frac{\nu\{\uparrow(\downarrow(\uparrow P \parr L) \otimes Q)\}}{\nu\{\uparrow(P \otimes Q) \parr L\}} \qquad \mathsf{ps_F}\,\frac{\nu\{\uparrow(P \otimes \downarrow(\uparrow Q \parr L))\}}{\nu\{\uparrow(P \otimes Q) \parr L\}} \qquad \mathsf{spr_F}\,\frac{\nu\{\uparrow!(N \parr ?P)\}}{\nu\{\uparrow!N \parr ?P\}}$$

If the premises and conclusions do not match (because of extra $\downarrow\uparrow$ pairs) we appeal to Lemma 23 and the remark above. ◀

▶ **Lemma 25.** *The rule $\mathsf{ps_F}$ is height-preserving admissible in LSS.*

Proof. Given a proof $\mathscr{D}$ of a negative formula N in $\mathsf{LSS} \cup \{\mathsf{ps_F}\}$, we prove by induction on the height of $\mathscr{D}$ that there is a proof of N in LSS of at most the same height as $\mathscr{D}$. In the base case, N is $\uparrow 1$ and we are done. In the general case, we case-analyze the bottommost rule instance r in $\mathscr{D}$. If this is not an instance of $\mathsf{ps_F}$, we appeal to the induction hypothesis on the proof above r and compose the result with r. In the case where r is an instance of $\mathsf{ps_F}$, we consider the rule instance r_1 above r in $\mathscr{D}$, and consider the cases for r_1. If r_1 is not a synthetic rule, then we can permute r up above r_1 and then appeal to the induction hypothesis on the proof now above r_1. If r_1 was an instance of $\mathsf{ct_F}$ or $\mathsf{dt_F}$, then we need to appeal to the induction hypothesis twice, which is possible because our reduction does not increase the height of $\mathscr{D}$. If r_1 was $\mathsf{wk_F}$ or $\mathsf{gc_F}$, we do not need to appeal to the induction hypothesis. If $\mathsf{r}_1 \in \{\mathsf{sai_F}, \mathsf{spc_F}, \mathsf{spr_F}\}$, we merge r and r_1 by replacing them with a new instance of r_1, as follows:

$$\begin{array}{ccc}
\begin{array}{c}\mathsf{sai_F}\,\dfrac{\nu\{\uparrow\pi_{\otimes}\{\downarrow\uparrow\rho_{\otimes}\{1\}\}\}}{\nu\{\uparrow\pi_{\otimes}\{\downarrow(\uparrow\rho_{\otimes}\{a\} \parr \overline{a})\}\}}\\ \mathsf{ps_F}\,\dfrac{}{\nu\{\uparrow\pi_{\otimes}\{\rho_{\otimes}\{a\}\} \parr \overline{a}\}}\end{array} &
\begin{array}{c}\mathsf{spc_F}\,\dfrac{\nu\{\uparrow\pi_{\otimes}\{\downarrow\uparrow\rho_{\otimes}\{\downarrow M \parr L\}\}\}}{\nu\{\uparrow\pi_{\otimes}\{\downarrow(\uparrow\rho_{\otimes}\{\downarrow M\} \parr L)\}\}}\\ \mathsf{ps_F}\,\dfrac{}{\nu\{\uparrow\pi_{\otimes}\{\rho_{\otimes}\{\downarrow M\}\} \parr L\}}\end{array} &
\begin{array}{c}\mathsf{spr_F}\,\dfrac{\nu\{\uparrow\pi_{\otimes}\{\downarrow\uparrow\rho_{\otimes}\{!(M \parr ?L)\}\}\}}{\nu\{\uparrow\pi_{\otimes}\{\downarrow(\uparrow\rho_{\otimes}\{!M\} \parr ?P)\}\}}\\ \mathsf{ps_F}\,\dfrac{}{\nu\{\uparrow\pi_{\otimes}\{\rho_{\otimes}\{!M\}\} \parr ?P\}}\end{array}\\
\downarrow & \downarrow & \downarrow\\
\mathsf{sai_F}\,\dfrac{\nu\{\uparrow\pi_{\otimes}\{\rho_{\otimes}\{1\}\}\}}{\nu\{\uparrow\pi_{\otimes}\{\rho_{\otimes}\{a\}\} \parr \overline{a}\}} &
\mathsf{spc_F}\,\dfrac{\nu\{\uparrow\pi_{\otimes}\{\rho_{\otimes}\{\downarrow M \parr L\}\}\}}{\nu\{\uparrow\pi_{\otimes}\{\rho_{\otimes}\{\downarrow M\}\} \parr L\}} &
\mathsf{spr_F}\,\dfrac{\nu\{\uparrow\pi_{\otimes}\{\rho_{\otimes}\{!(M \parr ?L)\}\}\}}{\nu\{\uparrow\pi_{\otimes}\{\rho_{\otimes}\{!M\}\} \parr ?P\}}
\end{array}$$

Now we appeal to the induction hypothesis on the proof above r_1 to produce a new proof on which we apply Lemma 23 to get a proof $\mathscr{D}'$, with a conclusion matching the premise of the new instances resulting from the merge. We appeal to the induction hypothesis again on $\mathscr{D}'$ and plug the result above the merged instance. Lastly, if r_1 is also an instance of $\mathsf{ps_F}$, then we appeal to the induction hypothesis on the proof above r_1 and apply the technique used for the other cases. ◀

We now have all the ingredients for the completeness theorem for LSS.

▶ **Theorem 26.** *For any N, if $\lfloor N \rfloor$ is provable in LS, then N is provable in LSS.*

Proof. Let a proof of $\lfloor N \rfloor$ in LS be given. By Lemma 22, there is a proof of $\lfloor N \rfloor$ in LS^{r}. By Lemma 24, we have a proof of N in $\mathsf{LSS} \cup \{\mathsf{ps_F}\}$, and thus by Lemma 25 also in LSS. ◀

Note that since LSS and LSF are equivalent, Theorem 26 also proves the completeness of LSF with respect to LS.

4 Comparing Sequent and Structural Focusing

In order to justify the adjective "focused" for LSF, it is important to give a precise comparison with LLKF. In this section we shall prove that every LLKF proof can be simulated in LSF, and, conversely, every LSF proof has a corresponding LLKF proof. Both results are surprising, as there is no reason *a priori* that the two systems should have such a close correspondence. Indeed, there are significant differences such as the treatment of weakening and contraction and the incremental splitting of contexts around $\otimes$.

4.1 Simulating LLKF in LSF

First, let us simulate LLKF proofs in LSF, *i.e.*, show that LSF is adequate with respect to LLKF. The two proof systems are not isomorphic, so we use an abstraction.

▶ **Definition 27.** For a non-empty LLKF sequent σ and a polarized formula A, we say that A *is a structural interpretation of* σ, written $A \approx \sigma$, iff it can be derived from these rules:

$$\frac{}{P \approx (\vdash \cdot \ ; \ [P])} \qquad \frac{Q \approx (\vdash \Gamma \ ; \ \Pi, [P])}{(Q \oplus L) \approx (\vdash \Gamma \ ; \ \Pi, L, [P])} \qquad \frac{Q \approx (\vdash \Gamma, P \ ; \ \Pi, [P'])}{(Q \oplus ?P) \approx (\vdash \Gamma, P \ ; \ \Pi, [P'])} \qquad \frac{Q \approx (\vdash \Gamma \ ; \ \Pi, [P'])}{Q \approx (\vdash \Gamma, P \ ; \ \Pi, [P'])}$$

$$\frac{}{N \approx (\vdash \cdot \ ; \ N)} \qquad \frac{M \approx (\vdash \Gamma \ ; \ \Delta)}{(M \parr N) \approx (\vdash \Gamma \ ; \ \Delta, N)} \qquad \frac{M \approx (\vdash \Gamma, P \ ; \ \Delta)}{(M \parr ?P) \approx (\vdash \Gamma, P \ ; \ \Delta)} \qquad \frac{M \approx (\vdash \Gamma \ ; \ \Delta)}{M \approx (\vdash \Gamma, P \ ; \ \Delta)}$$

In other words, structural interpretations can arbitrarily reorder the LLKF sequent and potentially erase or duplicate the unrestricted formulas, but they must preserve the multiplicities of the linear formulas. The simulation theorem shows that LSF can preserve the structural interpretations of each rule of LLKF.

▶ **Theorem 28** (simulation). *For any* Γ, Δ, Π, *and* P,

- *If* $\vdash \Gamma \ ; \ \Pi, [P]$ *in* LLKF, *then there is a* $Q \approx (\vdash \Gamma \ ; \ \Pi, [P])$ *such that* $\uparrow 1 \Vert_{\mathsf{LSF}\setminus\{\mathsf{ct_F}\}} Q$.
- *If* $\vdash \Gamma \ ; \ \Delta$ *in* LLKF, *then there is a* $N \approx (\vdash \Gamma \ ; \ \Delta)$ *such that* $1 \Vert_{\mathsf{LSF}\setminus\{\mathsf{ct_F}\}} N$.

Proof. By structural induction on the given LLKF proofs. ◀

▶ **Corollary 29** (completeness). *If* $\vdash P_1, \ldots, P_m \ ; \ N_1, \ldots, N_n$ *is provable in* LLKF, *then* $N_1 \parr \cdots \parr N_n \parr ?P_1 \parr \cdots \parr ?P_m$ *is provable in* LSF.

Proof. We have:

$$\begin{array}{c} \uparrow 1 \\ \Vert \text{ Theorem 28} \\ N_1 \parr \cdots \parr N_n \parr^{u_1} ?P_1 \parr^{u_2} \cdots \parr^{u_m} ?P_m \\ \mathsf{ct_F}, \mathsf{wk_F} \ \Vert \\ N_1 \parr \cdots \parr N_n \parr ?P_1 \parr \cdots \parr ?P_m \end{array}$$

where $M_1 \parr^u M_2$ stands for $M_1 \parr (\underbrace{M_2 \parr \cdots \parr M_2}_{u \text{ times}})$ if $u \geq 1$, and for M_1 if $u = 0$. ◀

4.2 Extracting LLKF Proofs from LSF Proofs

Let an LSF proof $\mathscr{D}$ with conclusion N_0 be given. We present here an algorithm that extracts an LLKF proof of $\vdash \cdot \ ; \ N_0$ that is unique up to rule permutations which are entirely confined to the negative phases, *i.e.*, the extraction does not make any essentially non-deterministic choices. We begin by *labelling* the active and reactive formulas in N_0, *i.e.*, we modify the grammar of formulas as follows:

$$\begin{array}{rcl} P,Q & ::= & a_u \mid {!_u}N \mid {\downarrow_u}N \mid P \otimes Q \mid \mathsf{1} \mid P \oplus Q \mid \mathsf{0} \mid P \mathbin{⫽} N \\ N,M & ::= & \overline{a}_u \mid {?_u}P \mid {\uparrow_u}P \mid N \mathbin{\&} M \mid \top \mid N \parr M \mid \bot \end{array}$$

We use $u, v, \ldots$ for labels drawn from some infinite set, and Λ for a multi-set of labels. We write L_u or R_u to denote that the (re)active formula L or R has label u. The rules of LSF (Figure 4) are modified to be label-sensitive. The key cases are as follows:

$$\mathsf{int_F}\,\frac{\nu\{{\uparrow_u}(P \mathbin{⫽} L_v)\}}{\nu\{{\uparrow_u}P \parr L_v\}} \qquad \mathsf{ai_F}\,\frac{\pi\{\mathsf{1}\}}{\pi\{a_u \mathbin{⫽} \overline{a}_v\}} \qquad \mathsf{pc_F}\,\frac{\pi\{{\downarrow_u}(N \parr L_v)\}}{\pi\{{\downarrow_u}N \mathbin{⫽} L_v\}} \qquad \mathsf{pr_F}\,\frac{\pi\{{!_u}\,(N \parr {?_v}P)\}}{\pi\{{!_u}N \mathbin{⫽} {?_v}P\}}$$

$$?_{\mathsf{F}}\,\frac{\nu\{{\uparrow_u}P\}}{\nu\{{?_u}P\}} \qquad \mathsf{ct_F}\,\frac{\nu\{{\uparrow_{u_1}}P \parr {\uparrow_{u_2}}P\}}{\nu\{{\uparrow_u}P\}}\,[\{u\} \rightsquigarrow \{u_1, u_2\}] \qquad \mathsf{wk_F}\,\frac{\nu\{\bot\}}{\nu\{{?_u}P\}}\,[\{u\} \rightsquigarrow \emptyset]$$

For all other rules the labelling is straightforward. The rules $\{\mathsf{int_F}, \mathsf{ai_F}, \mathsf{pr_F}, \mathsf{pc_F}\}$ in the first line above induce an ordering, written $<$, among the labels, with $u < v$ in each case. For $\mathsf{ct_F}$, the labels u_1 and u_2 in the premise are assumed to be different from each other and from all labels in the conclusion of the rule. The rules $\mathsf{ct_F}$ and $\mathsf{wk_F}$ induce a rewrite relation $\rightsquigarrow$ on multi-sets of labels that tracks the exponential uses of ?-formulas. We assume that if $u < v$ and $\{v\} \rightsquigarrow \Lambda, w$, then $u < w$. We label all active and reactive formulas in N_0 with unique labels and label every rule instance in $\mathscr{D}$ as above. Note that the reflexive-transitive closure of this label ordering, written $\leq$, is a partial order.

Our algorithm will extract a labelled LLKF proof of $\vdash \cdot\ ;\ N_0$ where the unrestricted contexts contain positive formulas annotated with a multi-set of labels, *i.e.*, their elements will be of the form P_Λ. The wdc rule is modified to consume one of the available labels in this multi-set; if this multi-set is empty, then the rule is inapplicable. Likewise, the dc rule consumes the label of the linear reactive formula. Finally, in the ? rule, the label of the ? is normalized with respect to $\rightsquigarrow$.

$$\mathsf{wdc}\,\frac{\vdash \Gamma, P_\Lambda\ ;\ \Pi, [P]}{\vdash \Gamma, P_{\Lambda,u}\ ;\ \Pi} \qquad \mathsf{dc}\,\frac{\vdash \Gamma\ ;\ \Pi, [P]}{\vdash \Gamma\ ;\ \Pi, {\uparrow_u}P} \qquad ?\,\frac{(\{u\} \Downarrow^{\rightsquigarrow} \Lambda) \quad \vdash \Gamma, P_\Lambda\ ;\ \Delta}{\vdash \Gamma\ ;\ \Delta, {?_u}P}$$

The remaining rules of LLKF (Figure 2) can be modified to use labelled formulas in a straightforward manner. We say that a label is *available* in a labelled LLKF sequent if it is the label of some top-level formula in the sequent. The extraction of the labelled LLKF proof of $\vdash \cdot\ ;\ N_0$ proceeds by backwards proof search (*i.e.*, proof search from this goal sequent upwards by applying LLKF rules from conclusion to premises) with some constraints:

- For a negative sequent for which there are available negative rules (*i.e.*, rules in the negative phase in Figure 2), we apply one of these rules. The choice and order of the application of these rules is immaterial.
- From a negative sequent where no negative rules apply, we pick the unique $\leq$-smallest label from the available labels, and use wdc or dc as appropriate. Each pair of formulas in such a sequent has a corresponding pair of formulas in a $\parr$-relation in $\mathscr{D}$. Because we normalize with respect to $\rightsquigarrow$, every surviving available label in this sequent is involved in some instance of $\mathsf{int_F}$ with another available label of the sequent in $\mathscr{D}$. Thus, the available labels are $\leq$-connected, and, because there is a sub-proof in $\mathscr{D}$ of the $\parr$-formula corresponding to this sequent, there is always a unique $\leq$-smallest label.
- For the $\otimes$ rule of LLKF focused on $P \otimes Q$, we send those side formulas to the premise involving P whose labels are $\leq$-larger than some label of a subformula of P, and the rest to the premise involving Q. There is no splitting ambiguity: the sets of labels in subformulas of P and Q are disjoint because the labelling is unique.
- For $\oplus$, we repeat the same choices made in $\mathscr{D}$.

This algorithm is deterministic and always succeeds: it has no choice points that require backtracking. There are no labels in any subformula of $\mathsf{1}$, so no formula will ever be sent to a branch of a $\otimes$ that has focus on $\mathsf{1}$, guaranteeing that its LLKF rule will succeed. Similarly, the sole formula that can be sent to a branch with focus on a_u will be a formula $\overline{a}_v$ with $u < v$, and therefore the id rule of LLKF will succeed. Lastly, the only formulas that have labels $\leq$-greater than a !-formula are ?-formulas, as ensured by the $\mathsf{pr_F}$ rule in LSF, so the corresponding ! rule of LLKF succeeds. The algorithm terminates because each decision rule consumes one of the finitely many labels of $\mathscr{D}$. We can erase the labels from the computed labelled LLKF proofs as a post-processing step.

References

1 Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992.

2 Arnon Avron. The method of hypersequents in the proof theory of propositional non-classical logics. In *Logic: from foundations to applications: European logic colloquium*, pages 1–32. Clarendon Press, 1996.

3 David Baelde and Dale Miller. Least and greatest fixed points in linear logic. In *LPAR*, volume 4790 of *LNCS*, pages 92–106, 2007.

4 Nuel D. Belnap, Jr. Display logic. *J. of Philosophical Logic*, 11:375–417, 1982.

5 Kai Brünnler. *Nested Sequents*. Habilitationsschrift, Universität Bern, 2010.

6 Kai Brünnler and Alwen Fernanto Tiu. A local system for classical logic. In *LPAR*, volume 2250 of *LNCS (LNAI)*, pages 347–361. Springer, 2001.

7 Paola Bruscoli and Alessio Guglielmi. On the proof complexity of deep inference. *ACM Trans. on Computational Logic*, 10(2), 2009.

8 Kaustuv Chaudhuri. Focusing strategies in the sequent calculus of synthetic connectives. In *LPAR*, volume 5330 of *LNCS*, pages 467–481, 2008.

9 Nicolas Guenot. Focused proof search for linear logic in the calculus of structures. In *ICLP*, volume 7 of *LIPIcs*, pages 84–93. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010.

10 Alessio Guglielmi. A system of interaction and structure. *ACM Trans. on Computational Logic*, 8(1), 2007.

11 Alessio Guglielmi and Lutz Straßburger. Non-commutativity and MELL in the calculus of structures. In *CSL*, volume 2142 of *LNCS*, pages 54–68. Springer, 2001.

12 Olivier Laurent. *Étude de la Polarisation en Logique*. PhD thesis, Aix-Marseille II, 2002.

13 Olivier Laurent. A proof of the focalization property of linear logic. Unpubl. note, 2004.

14 Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *TCS*, 410(46):4747–4768, 2009.

15 Dale Miller and Alexis Saurin. From proofs to focused proofs: a modular proof of focalization in linear logic. In *CSL*, volume 4646 of *LNCS*, pages 405–419. Springer, 2007.

16 P. O'Hearn and D. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

17 Lutz Straßburger. A local system for linear logic. In *LPAR*, volume 2514 of *LNCS (LNAI)*, pages 388–402. Springer, 2002.

18 Lutz Straßburger. *Linear Logic and Noncommutativity in the Calculus of Structures*. PhD thesis, Technische Universität Dresden, 2003.

19 Lutz Straßburger. MELL in the Calculus of Structures. *TCS*, 309(1–3):213–285, 2003.

20 Lutz Straßburger and Alessio Guglielmi. A system of interaction and structure IV: The exponentials and decomposition, 2009. To appear in ACM Trans. on Computational Logic.

A semantic approach to illative combinatory logic*

Łukasz Czajka

University of Warsaw
Institute of Informatics
lukaszczz@gmail.com

Abstract

This work introduces the theory of illative combinatory algebras, which is closely related to systems of illative combinatory logic. We thus provide a semantic interpretation for a formal framework in which both logic and computation may be expressed in a unified manner. Systems of illative combinatory logic consist of combinatory logic extended with constants and rules of inference intended to capture logical notions. Our theory does not correspond strictly to any traditional system, but draws inspiration from many. It differs from them in that it couples the notion of truth with the notion of equality between terms, which enables the use of logical formulas in conditional expressions. We give a consistency proof for first-order illative combinatory algebras. A complete embedding of classical predicate logic into our theory is also provided. The translation is very direct and natural.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Illative combinatory logic, term rewriting, first-order logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.174

1 Introduction

When in the early 1930s Curry and Church invented their systems of, respectively, combinatory logic [4] and lambda calculus [3], they intended them to be foundational systems on which logic and mathematics could be based. These systems were soon shown to be inconsistent by Kleene and Rosser [9]. As a result, Church abandoned his program of basing logic on lambda-calculus. Curry, however, persisted in his aims. He and his followers tried to formulate various systems weaker than the original system of Curry in the hope of obtaining ones that would be consistent, but still strong enough to interpret traditional logic. Today, the basic part of Curry's theory is known as combinatory logic, the systems additionally incorporating logical constants as illative combinatory logic. The search for strong and consistent theories proved to be elusive. Only after more than half a century since the first publications of Church and Curry, several systems were proven complete for minimal first-order intuitionistic logic in [2], [5], [6], and for PRED$\lambda\!\rightarrow$ in [5]. See [11] for a historical overview of illative combinatory logic.

The tradition of the Curry school has been formalist, with emphasis on constructive proof-theoretic methods (cf. [11]). In this work we propose a semantic interpretation for various illative constants. In contrast to traditional systems, the meaning of these constants is given by appropriately extending the equality relation. We attempt to give a model-theoretic style semantics. As potential models we study illative combinatory algebras. These are combinatory algebras with additional elements corresponding to illative constants. One important constant which is present in our theory, but usually absent from illative systems,

* Partly supported by MNiSW grant N N206 355836.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 174–188

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

is the conditional Cond. It acts as a connector between logic and computation, allowing to choose between two branches in a (generalized) program depending on the truth value of a quantified formula. Furthermore, formulas themselves are nothing else than generalized programs, and may contain S and K.

Our formalization is very natural and straightforward. What is non-obvious here is that it is actually correct. Modifying sligthly our axioms in seemingly harmless ways leads to inconsistent theories. In the presence of unrestricted abstraction and fixed points of arbitrary elements it is far from obvious how to formulate a consistent logical system.

Very closely related to our theory are applicative theories of Feferman (see [7]), which form the basis of his systems of explicit mathematics. These systems were intended to provide a foundation for constructive mathematics. Applicative theories are, however, usually based on partial logic. In terms of methods employed perhaps the total applicative theories with non-constructive μ-operator (see [8]) come even closer to our theory than does illative combinatory logic. Indeed, the key idea in the proofs leading to the central Corollary 42 is essentially analogous to that in the proof from the appendix of [8], where similar techniques are used in a much less general context. The author did not know about [8] until after having written down the proof in full.

Our consistency proof for first-order illative combinatory algebras is based on a non-trivial construction of a term model. We show how to extend any left-linear applicative term rewriting system satisfying some mild additional conditions into a term rewriting system whose associated quotient algebra is a first-order illative combinatory algebra. The extension is constructed by transfinitely iterating a process of expanding the term rewriting system with rules implementing quantification, until a fixpoint is reached. This bears some resemblance to transfinite truth definitions as used by Kripke (cf. [10]), which were also the inspiration for the three-valued semantics of logic programming. The details, however, are much more complicated.

The outline of the rest of this paper is as follows. Section 2 contains the definition of first-order illative combinatory algebras. In Section 3 we define a translation from first-order logic to illative language and prove its soundness. Section 4 introduces the class of functional term rewriting systems and recapitulates some known results from the theory of term rewriting. Section 5 contains the details of the term model construction. In Section 6 we use the result of Section 5 to prove completeness of the translation from Section 3.

2 Illative combinatory algebras

In this section we introduce the central concept of this work – illative combinatory algebras. Basic familiarity with ordinary combinatory logic is assumed.

▶ **Definition 1.** An *applicative algebra* $\mathcal{A}$ is a tuple $\langle \omega, \cdot, \upsilon \rangle$ where:

(1) ω is a set of *combinators*

(2) $\cdot : \omega \times \omega \to \omega$ is the *application function*

(3) $\upsilon \subseteq \omega$ is the set of *undefined* combinators

We call $\delta = \omega \setminus \upsilon$ the set of *defined* combinators. By $\omega(\mathcal{A})$, $\upsilon(\mathcal{A})$ we denote respectively the ω and υ components of $\mathcal{A}$, by $\delta(\mathcal{A})$ we denote $\omega(\mathcal{A}) \setminus \upsilon(\mathcal{A})$.

In expressions involving the application function we customarily omit parentheses and adopt the convention of association to the left, i.e. $M \cdot X \cdot Y \cdot Z$ stands for $((M \cdot X) \cdot Y) \cdot Z$. We will also sometimes omit the dots. We adopt the convention of referring to the elements of an algebra as *combinators*.

▶ **Definition 2.** An *illative combinatory algebra* (ICA) is an applicative algebra $\mathcal{A}$ with elements T, F, K, S, P, Q, Cond, A_δ which satisfy the following for any $X, Y, Z \in \omega(\mathcal{A})$:

(1) $T \neq F$

(2) $T,\ F \in \delta$

(3) $K \cdot X \cdot Y = X$

(4) $S \cdot X \cdot Y \cdot Z = X \cdot Z \cdot (Y \cdot Z)$

(5)
$$\begin{cases} P \cdot F \cdot X = T \\ P \cdot X \cdot T = T \\ P \cdot T \cdot F = F \\ P \cdot X \cdot Y \in \upsilon \text{ otherwise} \end{cases}$$

(6)
$$\begin{cases} \mathrm{Cond} \cdot T \cdot X \cdot Y = X \\ \mathrm{Cond} \cdot F \cdot X \cdot Y = Y \\ \mathrm{Cond} \cdot X \cdot Y \cdot Z \in \upsilon \text{ if } X \notin \{T, F\} \end{cases}$$

(7)
$$\begin{cases} Q \cdot X \cdot X \in \{T\} \cup \upsilon \\ Q \cdot X \cdot Y \in \{F\} \cup \upsilon \text{ for } X \neq Y \\ Q \cdot X \cdot Y \in \delta \text{ if } X, Y \in \delta \end{cases}$$

(8)
$$\begin{cases} A_\delta \cdot X = T \text{ if } X \in \delta \\ A_\delta \cdot X \in \{F\} \cup \upsilon \text{ if } X \in \upsilon \end{cases}$$

We will sometimes write A for A_δ.

▶ Remark. Intuitively, in an illative combinatory algebra undefined combinators are interpreted as meaningless at the object level, but not necessarily completely meaningless. Indeed, there may be undefined combinators which applied to a defined combinator give a defined result. The set δ is intuitively interpreted as the universe of discourse. It is intended to encompass everything we may meaningfully talk about at the object level. In particular, it includes the truth values T and F. The combinator A stands for a partial predicate which is true for elements of δ, and false or undefined for elements of υ. This predicate cannot be defined from the other combinators. The combinator Q is intended to represent a partial equality predicate.

▶ Remark. Any illative combinatory algebra satisfies the principle of combinatory abstraction and has a fixed point combinator. Thus, for every equation of the form

$$M \cdot X_1 \cdot \ldots \cdot X_n = \Phi(M, X_1, \ldots, X_n)$$

where $\Phi(Y, X_1, \ldots, X_n)$ is a combination of Y, X_1, …, X_n and some of the combinators postulated in the definition of an ICA, there exists a combinator M such that the equation holds for any combinators X_1, …, X_n. We will often rely on this fact and define combinators by such equations. Sometimes we will also use the lambda-notation $\lambda x\, \Phi(x)$ to denote a combinator M such that $MX = \Phi(X)$ for all $X \in \omega$. If there can be more than one combinator satisfying a given equation, then it is tacitly understood that we choose one such specific combinator and it does not matter which one.

▶ Remark. Our aim is to make as many combinators belong to δ as possible, since these are the combinators on which our additional elements are guaranteed to "work". However, one cannot get rid of υ altogether because the existence of fixed points of arbitrary combinators would lead to a contradiction. In fact, it can be easily shown that if $M \cdot X \in \delta$ for all $X \in \omega$ then $M \cdot X = M \cdot Y$ for all $X, Y \in \omega$.

For brevity, we will mostly omit explicit references to illative combinatory algebras. The following facts and definitions are to be understood that they are relative to some fixed illative combinatory algebra.

We use the notation N for $\lambda x.PxF$, $\wedge$ for $\lambda x.\lambda y.N(Px(Ny))$, and $\vee$ for $\lambda x.\lambda y.P(Nx)y$. We occasionally adopt infix notation for $\wedge$ and $\vee$.

It is easy to see that P, N, $\wedge$ and $\vee$ satisfy the following equations for any $X, Y \in \omega$:

$$\begin{aligned}
PXY = T &\quad \text{iff} \quad X = F \text{ or } Y = T\\
PXY = F &\quad \text{iff} \quad X = T \text{ and } Y = F\\
NX = T &\quad \text{iff} \quad X = F\\
NX = F &\quad \text{iff} \quad X = T\\
\wedge XY = T &\quad \text{iff} \quad X = T \text{ and } Y = T\\
\wedge XY = F &\quad \text{iff} \quad X = F \text{ or } Y = F\\
\vee XY = T &\quad \text{iff} \quad X = T \text{ or } Y = T\\
\vee XY = F &\quad \text{iff} \quad X = F \text{ and } Y = F
\end{aligned}$$

▶ **Definition 3.** A set of combinators $\tau \subseteq \omega$ is a *type* represented by $M \in \omega$ if the following conditions hold:

(1) $M \cdot X = T$ for $X \in \tau$
(2) $M \cdot X \in \{F\} \cup \upsilon$ for $X \in \omega \setminus \tau$

Note that ω and δ are types represented by $K \cdot T$ and A respectively. We use the notation b for the type represented by $A_b = \lambda x.(QxT) \vee (QxF)$.

▶ **Definition 4.** Let $\sigma, \rho \subseteq \omega$. A *function space* $\sigma \Rightarrow \rho$ from σ to ρ is the set of all combinators M such that $M \cdot X \in \rho$ for $X \in \sigma$.

We use small Greek letters τ, σ, ρ, ω, etc. both to denote subsets of ω and as parts of symbols denoting constants or combinators, e.g. in A_τ. In the second case the subscript does not have a meaning of its own, but only highlights a connection of the symbol with some set τ, which may even be defined only after introducing the symbol itself. Analogously, we use subscripts of the form $\sigma \to \rho$ when we intend to highlight a connection to the function space $\sigma \Rightarrow \rho$. In compound expressions $\to$ and $\Rightarrow$ are assumed to be right-associative. We adopt the notation $\sigma^n \Rightarrow \rho$ for $\sigma \Rightarrow \ldots \Rightarrow \sigma \Rightarrow \rho$ where σ occurs n times. Analogously, we use $\sigma^n \to \rho$ in subscripts.

▶ **Definition 5.** A combinator M is τ-total, for $\tau \subseteq \omega$, if $MX \in \delta$ for all $X \in \tau$.

▶ **Definition 6.** Let $\tau \subseteq \omega$. A *τ-quantifier* is any combinator Π_τ such that:

$$\begin{aligned}
\Pi_\tau \cdot X &= T \text{ if for all } Y \in \tau \text{ we have } X \cdot Y = T\\
\Pi_\tau \cdot X &= F \text{ if there exists } Y \in \tau \text{ such that } X \cdot Y = F\\
\Pi_\tau \cdot X &\in \upsilon \text{ otherwise}
\end{aligned}$$

We use the notation Σ_τ for $\lambda x.N(\Pi_\tau(S(KN)x))$. A combinator Π_δ satisfying the above equations for $\tau = \delta$ is a *first-order quantifier*. We will sometimes write Π instead of Π_δ.

It is straightforward to verify that Π_τ and Σ_τ satisfy the following for any $X \in \omega$:

$$\begin{aligned}
\Pi_\tau X = T &\quad \text{iff} \quad XY = T \text{ for all } Y \in \tau\\
\Pi_\tau X = F &\quad \text{iff} \quad XY = F \text{ for some } Y \in \tau\\
\Sigma_\tau X = T &\quad \text{iff} \quad XY = T \text{ for some } Y \in \tau\\
\Sigma_\tau X = F &\quad \text{iff} \quad XY = F \text{ for all } Y \in \tau
\end{aligned}$$

It is easy to see that if A_τ is a δ-total combinator representing a type $\tau \subseteq \delta$, then the combinator $\lambda x.\Pi_\delta \lambda y.P\,(A_\tau y)\,(xy)$ is a τ-quantifier. Moreover, if Π_{τ_1} is a τ_1-quantifier and A_{τ_2} represents a type τ_2, then $\tau_1 \Rightarrow \tau_2$ is a type represented by $A_{\tau_1 \to \tau_2} = \lambda x.\Pi_{\tau_1} \lambda y.A_{\tau_2}(xy)$.

▶ **Definition 7.** A *first-order illative combinatory algebra* (FO-ICA) is an illative combinatory algebra with signature extended with Π_δ, and with the laws from Definition 6 for Π_δ added as axioms.

▶ Remark. One may wonder why we postulate the existence of Π_δ instead of Π_ω, whose range of quantification is broader. After all, we could use $\Pi'_\delta = \lambda x.\Pi_\omega \lambda y.P(Ay)(xy)$. However, Π'_δ is not a δ-quantifier. The reason is the existence of undefined combinators and the fact that they are included in the range of quantification of Π_ω. For instance, suppose M is such that $M \cdot X = T$ iff $X \in \delta$. One may easily show that there is $Y \in \upsilon$ such that $A \cdot Y \in \upsilon$. Hence, $P(AY)(MY) \in \upsilon$. Moreover, by definitions of A and M there is no Z such that $P(AZ)(MZ) = F$. So the last equation in the definition of Π_ω applies, and we have $\Pi'_\delta M \in \upsilon$.

More generally, if A_τ represents a type $\tau \neq \omega$, then by an analogous argument we could prove that $\Pi_\omega \lambda x.P(A_\tau x)(Mx) \in \upsilon$ for any $M \in \omega$ such that $M \cdot X = T$ iff $X \in \tau$. This shows that Π_ω is not particularly interesting, because its range cannot be restricted in a meaningful way.

▶ Remark. Logic based on the theory of first-order illative combinatory algebras is, in a practical sense, more expressive than traditional predicate logic. For example, denote by $\underline{n}$ the Church numeral representing $n \in \mathbb{N}$. Now we can write a recursive definition of U as follows:

$$U\underline{n} \quad = \quad \mathrm{Cond}\,(Q\,\underline{n}\,0)\,(SKK)\;(\lambda f.\Pi\lambda x.\Sigma\lambda y.U(\mathrm{Pred}\,\underline{n})(fxy))$$

where Pred is the predecessor combinator for Church numerals. By simple induction one can show:

$$U\underline{n} \quad = \quad \lambda f.\Pi\lambda x_1.\Sigma\lambda y_1.\ldots\Pi\lambda x_n.\Sigma\lambda y_n.fx_1y_1x_2y_2\ldots x_ny_n$$

Now assume that we have a δ-total combinator which represents the type $\mathcal{N}$ consisting of Church numerals, and that all Church numerals are in δ. Theorem 43 implies that the definition of a FO-ICA may be modified to satisfy these assumptions without sacrificing any of the results in this paper. Then there exists an $\mathcal{N}$-quantifier $\Pi_\mathcal{N}$. Now, given a combinator M, the expression

$$\Sigma_\mathcal{N}\lambda x.UxM$$

is true iff there exists an alternation of $2n$ quantifiers such that

$$\Pi\lambda x_1.\Sigma\lambda y_1.\ldots\Pi\lambda x_n.\Sigma\lambda y_n.Mx_1y_1\ldots x_ny_n$$

is true. To be precise, $\Sigma_\mathcal{N}\lambda x.UxM$ will most often be in υ if such an alternation does not exist.

The power comes from the fact that quantifiers may be freely combined with S and K. This allows for recursive definitions involving logical operators.

Another important feature of our theory is the presence of the combinator Cond and the fact that the truth notion at the meta-level is coupled with the notion of equality between terms. In other words, being true is equivalent to evaluating to a concrete value T, which may be used in the "program" itself. This is significantly different from simply stating that

some terms are "true" or "derivable" by means of some meta-level definition, but without providing any possibility of using this information *inside* the system.

For instance, with our approach one can write recursive definitions of the form:

$$M \quad = \quad \lambda x.\Psi\left(\mathrm{Cond}\left(\Pi\Phi_1(x,M)\right)\Phi_2(x,M)\,\Phi_3(x,M)\right)$$

and they behave as expected – if $\Pi\Phi_1(X,M)$ is true then the first branch constitutes the value of MX, if false then the second. What is more, it may happen that we know that $\Pi\Phi_1(X,M)$ is true regardless of what X is, and we may conclude that $M = \lambda x.\Psi\left(\Phi_2(x,M)\right)$. The combinator Cond acts as a connector between logic and computation.

3 Translation from first-order to illative theories

In this section we define a natural translation from the language of first-order logic to illative language and prove its soundness with respect to FO-ICAs. We defer the proof of completeness to Section 6. Much of the present section contains some fairly obvious but necessary definitions.

We will be dealing mostly with *applicative* terms, i.e. terms from languages over signatures consisting solely of a single binary function symbol $\cdot$ and constants including all the constants postulated in the definition of a FO-ICA. We denote such a language by $\mathcal{L}(\Sigma, V)$, where Σ is a set of constants, and V is a set of variables. All terms are assumed to be applicative, unless qualified with the phrase *first-order*. We use the symbols t, s, etc. for terms, x, y, etc. for variables, and M, X, etc. for combinators (elements of an algebra), except that we use the same symbols for primitive constants and corresponding combinators defined in Section 2. The intended meaning of a symbol will always be clear from the context.

We use the notation $[\![t]\!]^u_{\mathcal{A}}$ for the value of t under variable valuation u in the structure $\mathcal{A}$. We omit the decorations when obvious from the context or irrelevant. We also adopt the notation $t_1[x/t_2]$ for the term t_1 with all free occurences of x substituted for t_2. Analogously, we use $u[x/M]$ for the valuation u' such that $u'(y) = u(y)$ for $y \neq x$ and $u'(x) = M$.

We define lambda-abstraction at the syntactic level by the standard abstraction algorithm: $\lambda^* x.x = SKK$, $\lambda^* x.t = Kt$ if $x \notin FV(t)$, and $\lambda^* x.t_1 t_2 = S(\lambda^* x.t_1)(\lambda^* x.t_2)$. In what follows the symbols A_b, $\wedge$, etc. will sometimes stand for terms defined completely analogously to the corresponding combinators in Section 2, but at the syntactic level using the abstraction algorithm. We still use these symbols to denote the combinators as well. Again, the intended meaning will always be clear from the context.

Let $\mathcal{A}$ be a FO-ICA, and u a valuation. It is easy to verify that for any terms t_1, t_2 we have $[\![(\lambda^* x.t_1)t_2]\!]^u_{\mathcal{A}} = [\![t_1[x/t_2]]\!]^u_{\mathcal{A}}$. Also for any term t and any $M \in \omega(\mathcal{A})$ we have the identity $[\![\lambda^* x.t_1]\!]^u_{\mathcal{A}} \cdot M = [\![t_1]\!]^{u'}_{\mathcal{A}}$ where $u' = u[x/M]$.

We now redefine some standard notions from elementary first-order logic. Subseqently, we will refer to the original notions by qualifying them with the phrase *first-order*. The redefined notions will be qualified with *illative*, but the qualification will often be dropped. By an *illative theory* we mean a set of applicative terms. We say that a FO-ICA $\mathcal{A}$ *satisfies* a term t under variable valuation u, denoted by $\mathcal{A} \models^u t$, if $[\![t]\!]^u_{\mathcal{A}} = T$. We define the notions of *illative semantic consequence* ($\Gamma \models t$) and *illative model* ($\mathcal{A} \models \Gamma$) completely analogously to standard definitions in first-order logic, but with arbitrary terms in place of formulas and requiring all structures to be FO-ICAs.

We use the symbol Δ for a first-order theory, ϕ, ψ for first-order formulas, $\models_{FO}$ for the first-order semantic consequence relation.

By a *first-order expression* we mean a first-order formula or a first-order term. We extend the notion of first-order valuation to formulas. If $\mathcal{A} \models^u_{FO} \phi$ then $[\![\phi]\!]^u_{\mathcal{A}} = T$, otherwise $[\![\phi]\!]^u_{\mathcal{A}} = F$.

We assume that in a first-order language the only logical connective is $\rightarrow$, the only quantifier $\forall$, and there is a constant $\bot$ for false. We also assume that we have a new constant A_ς in the illative signature, and the signature contains as constants all symbols (of any arity) from the corresponding first-order language.

We write A_ι for the term $\lambda^* x.(A_\varsigma x) \wedge (A_\delta x)$ and Π_ι for $\lambda^* y.\Pi_\delta \lambda^* x.P(A_\iota x)(yx)$. We define $A_{\iota^{n+1}\rightarrow\iota}$ inductively as $\lambda^* x.\Pi_\iota \lambda^* y.A_{\iota^n\rightarrow\iota}(xy)$, where $A_{\iota^0\rightarrow\iota} = A_\iota$. Analogously, we define $A_{\iota^{n+1}\rightarrow b}$ as $\lambda^* x.\Pi_\iota \lambda^* y.A_{\iota^n\rightarrow b}(xy)$.

▶ **Definition 8.** The illative theory Γ_0 constains the terms $\Pi_\delta\,(S\,(K A_b)\,A_\iota)$, $\Sigma_\delta A_\iota$, $A_{\iota^n\rightarrow\iota}\,f$ for all function symbols f of arity $n \geq 0$, and $A_{\iota^n\rightarrow b}\,r$ for all relation symbols r of arity $n > 0$.

▶ **Definition 9.** We define inductively a translation Ψ as follows.

For first-order terms:

- $\Psi(x) = x$ for a variable x
- $\Psi(f(t_1, \ldots, t_n)) = f \cdot \Psi(t_1) \cdot \ldots \cdot \Psi(t_n)$ for a function symbol f of arity $n \geq 0$

For first-order formulas:

- $\Psi(\bot) = F$
- $\Psi(r(t_1, \ldots, t_n)) = r \cdot \Psi(t_1) \cdot \ldots \cdot \Psi(t_n)$ for a relation symbol r of arity n
- $\Psi(t_1 = t_2) = Q \cdot \Psi(t_1) \cdot \Psi(t_2)$
- $\Psi(\phi_1 \rightarrow \phi_2) = P \cdot \Psi(\phi_1) \cdot \Psi(\phi_2)$
- $\Psi(\forall_x \phi) = \Pi_\iota \lambda^* x.\Psi(\phi)$

For a first-order theory Δ we define $\Psi(\Delta)$ to be the sum of Γ_0 and the image $\mathrm{Im}_\Psi(\Delta)$ of Ψ on Δ.

The general idea of the soundness proof is to construct for every illative model $\mathcal{B}$ of $\Psi(\Delta)$ a first-order structure $\mathcal{A}$ which satisfies exactly those sentences whose translations are true in $\mathcal{B}$. Such a structure will obviously be a model of Δ. Hence, any semantic consequence ϕ of Δ will be satisfied by $\mathcal{A}$, so $\Psi(\phi)$ will be true in $\mathcal{B}$.

▶ **Definition 10.** Let $\mathcal{B}$ be an illative model of Γ_0. We define $\iota_\mathcal{B}$ to be the set of all combinators $M \in \omega(\mathcal{B})$ such that $[\![A_\iota]\!]_\mathcal{B} \cdot M = T$. The subscript will be dropped when obvious from the context.

It is easy to see that if $\mathcal{B}$ is an illative model of Γ_0, then $[\![A_\iota]\!]_\mathcal{B}$ is a δ-total combinator representing the non-empty type $\iota_\mathcal{B} \subseteq \delta(\mathcal{B})$. It is also true that in every illative model $\mathcal{B}$ of Γ_0 the combinator $[\![\Pi_\iota]\!]_\mathcal{B}$ is a $\iota_\mathcal{B}$-quantifier.

▶ **Definition 11.** A first-order structure $\mathcal{A}$ and a FO-ICA $\mathcal{B}$ are *correspondent* if the universe U of $\mathcal{A}$ is a subset of $\omega(\mathcal{B})$ and the following conditions hold:

- every function symbol f of arity n is interpreted in $\mathcal{A}$ by the function

$$\{(X_1, \ldots, X_n, Y) \in U^{n+1} \mid [\![f]\!]_\mathcal{B} \cdot X_1 \cdot \ldots \cdot X_n = Y\}$$

- every relation symbol r of arity n is interpreted in $\mathcal{A}$ by the relation

$$\{(X_1, \ldots, X_n) \in U^n \mid [\![r]\!]_\mathcal{B} \cdot X_1 \cdot \ldots \cdot X_n = T\}$$

▶ **Lemma 12.** *Assume $\mathcal{B}$ is an illative model of Γ_0 and $\mathcal{A}$ is a first-order structure with $\iota_{\mathcal{B}}$ as the universe. If $\mathcal{A}$ and $\mathcal{B}$ are correspondent, then $[\![e]\!]^u_{\mathcal{A}} = [\![\Psi(e)]\!]^u_{\mathcal{B}}$ for any first-order expression e and any valuation u such that $Rg(u) \subseteq \iota_{\mathcal{B}}$.*

Proof. Simple induction on the complexity of e.

For instance, assume $e = \forall_x \psi$ and $[\![\forall_x \psi]\!]^u_{\mathcal{A}} = T$. Then for every $X \in \iota$ we have $T = [\![\psi]\!]^{u'}_{\mathcal{A}} = [\![\Psi(\psi)]\!]^{u'}_{\mathcal{B}}$ by inductive hypothesis, where $u' = u[x/X]$. So for every $X \in \iota$ we have $[\![\lambda^* x.\Psi(\psi)]\!]^u_{\mathcal{B}} \cdot X = [\![\Psi(\psi)]\!]^{u'}_{\mathcal{B}} = T$. Hence, by the fact that $[\![\Pi_\iota]\!]_{\mathcal{B}}$ is a ι-quantifier in $\mathcal{B}$ we have $[\![\Pi_\iota \lambda^* x.\Psi(\psi)]\!]^u_{\mathcal{B}} = T$.

The other cases are similar. We need the assumption of correspondence in the cases $e = f(t_1, \ldots, t_n)$ and $e = r(t_1, \ldots, t_n)$. In the second of these $A_{\iota^n \to b}\, r \in \Gamma_0$ is also needed. ◀

▶ **Theorem 13. Soundness**
Let ϕ and all formulas in Δ be closed. If $\Delta \models_{FO} \phi$ then $\Psi(\Delta) \models \Psi(\phi)$.

Proof. Suppose $\Delta \models_{FO} \phi$. Because all terms in $\Psi(\Delta)$ as well as $\Psi(\phi)$ are closed by our construction, it suffices to show that every illative model of $\Psi(\Delta)$ is an illative model of $\Psi(\phi)$.

So assume $\mathcal{B}$ is an illative model of $\Psi(\Delta)$. Since $\Gamma_0 \subseteq \Psi(\Delta)$ then $\mathcal{B}$ is an illative model of Γ_0. Hence, $\iota_{\mathcal{B}} \subseteq \delta(\mathcal{B})$ is a non-empty type represented by $[\![A_\iota]\!]_{\mathcal{B}}$.

Let $\mathcal{A}$ be a first-order structure with universe ι and functions and relations as in Definition 11. Note that it is not immediately obvious that this is well-defined, because the interpretation of a function symbol f of arity n must be a total function from ι^n to ι. However, this is satisfied because $A_{\iota^n \to \iota}\, f \in \Psi(\Delta)$. Note also that the non-emptiness of ι is necessary because the universe of a first-order structure is always assumed to be non-empty.

Therefore, by Lemma 12 we may conclude that $[\![\psi]\!]_{\mathcal{A}} = [\![\Psi(\psi)]\!]_{\mathcal{B}}$ for every closed first-order formula ψ. We have $\mathcal{A} \models_{FO} \Delta$, because $[\![\psi]\!]_{\mathcal{A}} = [\![\Psi(\psi)]\!]_{\mathcal{B}} = T$ for $\psi \in \Delta$. From the initial assumption $\Delta \models_{FO} \phi$ we may now conclude that $\mathcal{A} \models_{FO} \phi$. This implies $[\![\Psi(\phi)]\!]_{\mathcal{B}} = [\![\phi]\!]_{\mathcal{A}} = T$. Therefore, $\mathcal{B} \models \Psi(\phi)$. ◀

4 Functional term rewriting systems

This section defines the class of functional term rewriting systems and briefly recapitulates some known results from the term rewriting theory for the sake of completeness. Term rewriting notation and terminology conforms to that from [1].

▶ **Definition 14.** The set of *positions* of a term $t \in \mathcal{L}(\Sigma, V)$ is a set $\mathrm{Pos}(t)$ of strings over the alphabet $\{0, 1\}$ defined inductively as follows: $\mathrm{Pos}(t_0 \cdot t_1) = \{\varepsilon\} \cup \{0p \mid p \in \mathrm{Pos}(t_0)\} \cup \{1p \mid p \in \mathrm{Pos}(t_1)\}$, and $\mathrm{Pos}(x) = \varepsilon$, where ε is the empty string and $x \in V$. The *leftmost position* of t is the position 0^i, where 0^n for $n \in \mathbb{N}$ means 0 repeated n times, such that no position of t is of the form 0^j for $j > i$. For $p \in \mathrm{Pos}(t)$, the *subterm of s at position p*, denoted by $t_{|p}$, is defined by induction on the length of p: $t_{|\varepsilon} = t$, $(t_0 \cdot t_1)_{|bq} = t_{b|q}$. A *context* C is a term over $\mathcal{L}(\Sigma \cup \{\Box\}, V)$ with exactly one occurence of $\Box$. By $C[t]$ for $t \in \mathcal{L}(\Sigma, V)$ we denote the term C with $\Box$ replaced by t.

▶ **Definition 15.** A *rewrite rule*, or simply *rule*, over $\mathcal{L}(\Sigma, V)$ is a pair $(l, r) \in \mathcal{L}(\Sigma, V) \times \mathcal{L}(\Sigma, V)$ such that l is not a variable and $\mathrm{Var}(r) \subseteq \mathrm{Var}(l)$. Rewrite rules will be written as $l \to r$. The term l is called the *left side* of the rule, r the *right side*. A rule $l \to r$ is *left-linear* if no variable occurs twice in l. A rule $l \to r$ is *trivial* if $l = r$. A *term rewriting system* is a set of rewrite rules. A term rewriting system is *left-linear* if each of its rules is left-linear.

Let R be a term rewriting system. The *reduction relation* $\to_R \subseteq \mathcal{L}(\Sigma, V) \times \mathcal{L}(\Sigma, V)$ is defined as follows:

$$t \to_R s \quad \text{iff} \quad \begin{array}{l}\text{there exist } l \to r \in R, \text{ a context } C \text{ and a substitution } \sigma \\ \text{such that } t = C[\sigma l] \text{ and } s = C[\sigma r].\end{array}$$

We sometimes write $t \to_R^p s$ to indicate at which position the reduction takes place.

We say that a rule $l \to r \in R$ *applies* to t if there exist $p \in \mathrm{Pos}(t)$ and a substitution σ such that $t_{|p} = \sigma l$. We say that a term is in *R-normal form* if no rule from R applies to it.

▶ **Notation 16.** Let $\to$ be a binary relation on terms. We denote by $\to^{\equiv}$ the reflexive closure of $\to$, by $\xrightarrow{*}$ the reflexive transitive closure, and by $\overset{*}{\leftrightarrow}$ the reflexive transitive symmetric closure. We write $t \to s$ to indicate that $(t, s) \in \to$. Analogously for $\to^{\equiv}$, $\xrightarrow{*}$ and $\overset{*}{\leftrightarrow}$.

▶ **Definition 17.** A position p of a term $t \in \mathcal{L}(\Sigma, V)$ is a *function position* if either $p = q0$ or the size of t is 1 and $p = \varepsilon$. A term $t \in \mathcal{L}(\Sigma, V)$ is *functional* if it does not have any variables at function positions. We use the notation $\Sigma_f(t)$ for the set of constants at function positions in a term t. By $\mathcal{H}(t)$ we denote the constant at the leftmost position in a functional term t. A rule $l \to r$ is *functional* if l is a functional term. A *functional term rewriting system* (FTRS) is a term rewriting system over $\mathcal{L}(\Sigma, V)$, such that all rules are functional. We use the notation $\Sigma_f(R)$ for $\bigcup_{l \to r \in R} \Sigma_f(l)$, and $\mathcal{H}(R)$ for $\{\mathcal{H}(l) \mid l \to r \in R\}$.

▶ **Fact 18.** *If t is a functional term and s is such that $\Sigma_f(t) \not\subseteq \Sigma_f(s)$, then there is no substitution σ and position p such that $s_{|p} = \sigma t$. Moreover, if s is a functional term and $\mathcal{H}(t) \notin \Sigma_f(s)$, then t does not unify with a non-variable subterm of s.*

▶ **Definition 19.** A functional term rewriting system R *generates* an applicative algebra $\mathcal{A}_R = \langle \omega, \cdot, \upsilon \rangle$ where $\omega = \{[t]_R\}$ is the set of equivalence classes of $\overset{*}{\leftrightarrow}_R$ on closed terms, υ is the set of those $[t]_R$ for which there is no t' in R-normal form such that $t \overset{*}{\leftrightarrow}_R t'$, and $\cdot$ is defined by $[t_1]_R \cdot [t_2]_R = [t_1 \cdot t_2]_R$.

▶ **Definition 20.** Let $l_1 \to r_1 \in R_1, l_2 \to r_2 \in R_2$, let p be a position such that $l_{1|p}$ is not a variable. We assume the rules do not share variable names. Let σ be the most general unifier of $l_{1|p}$ and l_2. Then $\langle \sigma r_1, (\sigma l_1)[\sigma r_2]_p \rangle$ is a *critical pair* between R_1 and R_2. A critical pair is a *root critical pair* if $p = \varepsilon$. The set of all critical pairs between R_1 and R_2 is denoted by $\mathrm{Crit}(R_1, R_2)$, the set of all root critical pairs by $\mathrm{Crit}_r(R_1, R_2)$, and $\mathrm{Crit}_i(R_1, R_2)$ is the set of all non-root critical pairs between R_1 and R_2. A critical pair $\langle u_1, u_2 \rangle \in \mathrm{Crit}(R_1, R_2)$ may be *closed* if $u_1 \to_{R_2} u_2$ or $u_2 \to_{R_1} u_1$.

▶ **Definition 21.** R_1 is *compatible* with R_2 if

- for all $\langle u_1, u_2 \rangle \in \mathrm{Crit}(R_1, R_2)$ there is u such that $u_1 \xrightarrow{*}_{R_2} u$ and $u_2 \to^{\equiv}_{R_1} u$,
- for all $\langle u_2, u_1 \rangle \in \mathrm{Crit}_i(R_2, R_1)$ we have $u_1 \to^{\equiv}_{R_2} u_2$.

Two term rewriting systems R_1, R_2 are compatible if R_1 is compatible with R_2 or *vice versa*.

▶ **Fact 22.** *If R_1, R_2 are FTRSes such that $\Sigma_f(R_1) \cap \Sigma_f(R_2) = \emptyset$, then they are compatible.*

▶ **Definition 23.** We say that two relations $\to_1$ and $\to_2$ *commute* whenever $t \xrightarrow{*}_1 t_1$ and $t \xrightarrow{*}_2 t_2$ implies the existence of s such that $t_1 \xrightarrow{*}_2 s$ and $t_2 \xrightarrow{*}_1 s$. Two term rewriting systems R_1 and R_2 commute if $\to_{R_1}$ and $\to_{R_2}$ commute.

▶ **Lemma 24. Commutative Union Lemma**
If R_1 and R_2 are confluent and they commute, then $R_1 \cup R_2$ is confluent.

The following theorem is a special case of the result from [12]. We do not state it in its full generality mostly due to lack of space to introduce the necessary concepts.

▶ **Theorem 25.** *Compatible left-linear term rewriting systems commute.*

It follows from Theorem 25 and the Commutative Union Lemma that if R is left-linear and compatible with itself, then it is confluent.

5 Extensions of standard systems

This section contains the mathematically non-trivial part of this work. We show how to extend any FTRS satisfying some mild additional conditions into an FTRS that generates a FO-ICA.

▶ **Definition 26.** A functional term rewriting system R is *standard* if it is left-linear, confluent, and $\Sigma_f(R) \cap \{\Pi, Q, A\} = \emptyset$.

▶ **Definition 27.** The term rewriting system PROP is defined by the following rules:

$$\begin{aligned}
K \cdot x \cdot y &\to x && (1)\\
S \cdot x \cdot y \cdot z &\to x \cdot z \cdot (y \cdot z) && (2)\\
P \cdot F \cdot x &\to T && (3)\\
P \cdot x \cdot T &\to T && (4)\\
P \cdot T \cdot F &\to F && (5)\\
P \cdot x \cdot y &\to P \cdot x \cdot y && (6)\\
\mathrm{Cond} \cdot T \cdot x \cdot y &\to x && (7)\\
\mathrm{Cond} \cdot F \cdot x \cdot y &\to y && (8)\\
\mathrm{Cond} \cdot x \cdot y \cdot z &\to \mathrm{Cond} \cdot x \cdot y \cdot z && (9)
\end{aligned}$$

▶ **Lemma 28.** *The term rewriting system PROP is standard.*

Proof. There are only the following root critical pairs, all of which satisfy the requirements of compatibility.

- The pair $\langle T, P \cdot F \cdot x \rangle$ between rules (3) and (6). We have $P \cdot F \cdot x \to T$ by rule (3).
- The root critical pairs between rules (6) and (3), (4) and (6), (6) and (4), (5) and (6), (6) and (5), (7) and (9), (9) and (7), (8) and (9), (9) and (8) are dealt with completely analogously to the above one.
- The trivial critical pair $\langle T, T \rangle$ between rules (3) and (4) or (4) and (3).

◀

Note that it follows directly from Lemma 28, Theorem 25 and the Commutative Union Lemma that $R \cup \mathrm{PROP}$ is standard.

▶ **Definition 29.** A term t is *R-standard* if it is a closed term in $R \cup \mathrm{PROP}$-normal form such that $\Sigma_f(t) \subseteq \Sigma_f(R \cup \mathrm{PROP})$.

For the rest of this section we assume a fixed standard functional term rewriting system R compatible with PROP, and a fixed family $\mathcal{T}_{\mathbb{I}} = \{\mathcal{T}_i \mid i \in \mathbb{I}\}$ of sets of R-standard terms, where $\mathbb{I}$ is some arbitrary index set.

▶ **Definition 30.** The term rewriting system R_I is defined by the following rules:

$$\begin{aligned}
\Pi \cdot x &\to \Pi \cdot x\\
Q \cdot x \cdot y &\to Q \cdot x \cdot y\\
A \cdot x &\to A \cdot x\\
A_{\mathcal{T}_i} \cdot t_i &\to T\\
A_{\mathcal{T}_i} \cdot x &\to A_{\mathcal{T}_i} \cdot x
\end{aligned}$$

for all $i \in \mathbb{I}$ and all terms $t_i \in \mathcal{T}_i$, where $A_{\mathcal{T}_i}$ are new symbols not present in $\Sigma_f(R) \cup \Sigma_f(\mathrm{PROP}) \cup \{\Pi, Q, A\}$.

It is easy to see that every R-standard term is in R_I-normal form.

▶ **Definition 31.** The term rewriting system R_{II} is defined by the following rules:

$$\begin{aligned} Q \cdot t \cdot t &\to T \\ Q \cdot t_1 \cdot t_2 &\to F \\ A \cdot t &\to T \\ A_{\mathcal{T}_i} \cdot t_i &\to F \end{aligned}$$

for all $i \in \mathbb{I}$ and all closed terms t, t_1, t_2, t_i in $R \cup \mathrm{PROP} \cup R_I$-normal form, such that $t_1 \neq t_2$ and $t_i \notin \mathcal{T}_i$.

Below we use the notation R_0 for $R \cup \mathrm{PROP} \cup R_I \cup R_{II}$.

▶ **Lemma 32.** *The term rewriting system R_0 is left-linear and confluent.*

Proof. It is evident that R_0 is left-linear. Notice that $R \cup \mathrm{PROP} \cup R_I$ is confluent because $R \cup \mathrm{PROP}$ and R_I are compatible by Fact 22 and the fact that $R \cup \mathrm{PROP}$ is standard.

We now prove that R_{II} is confluent. It is evident from Definition 31 that there are no root critical pairs. For two rules to form a non-root critical pair the left side of one of the rules has to unify with a proper subterm of the left side of the other rule. Because all left sides are closed terms, this is equivalent to the situation when the left side of one rule is equal to a proper subterm of another. It follows directly from definitions that all proper subterms of left sides of rules are in $R \cup \mathrm{PROP} \cup R_I$-normal form. However, no left side of a rule is in $R \cup \mathrm{PROP} \cup R_I$-normal form because the corresponding trivial rule from R_I applies. This implies that there are no critical pairs.

We show that $R \cup \mathrm{PROP} \cup R_I$ is compatible with R_{II}. Because $R \cup \mathrm{PROP}$ is standard, then by Fact 22 we need to consider only critical pairs between R_I and R_{II}. It is evident that all such pairs are root critical pairs between a trivial rule from R_I and a rule from R_{II}. They may be closed by simply applying the rule from R_{II}. ◀

▶ **Definition 33.** For an ordinal $\alpha > 0$, define R_α as the sum of $\bigcup_{\beta<\alpha} R_\beta$ and the rules:

- $\Pi \cdot t \to T$ for all closed terms t such that for any closed term s in R_0-normal form there is an ordinal $\beta < \alpha$ for which $t \cdot s \xrightarrow{*}_{R_\beta} T$.
- $\Pi \cdot t \to F$ for all closed terms t such that there is a closed term s in R_0-normal form and an ordinal $\beta < \alpha$ for which $t \cdot s \xrightarrow{*}_{R_\beta} F$.

A simple cardinality argument shows that there exists the least ordinal ζ such that $R_\zeta = \bigcup_{\alpha<\zeta} R_\alpha$. We sometimes write $R_\zeta^{\mathcal{T}_\mathbb{I}}$ for R_ζ when $\mathcal{T}_\mathbb{I}$ is not obvious from the context.

▶ **Lemma 34.** *For $\alpha \geq 0$, a term t is in $R \cup PROP \cup R_I$-normal form iff it is in R_α-normal form.*

Proof. If a rule from $R_\alpha \setminus (R \cup \mathrm{PROP} \cup R_I)$, e.g. $Q \cdot s \cdot s \to T$, applies to t, then the corresponding trivial rule, e.g. $Q \cdot x \cdot x \to Q \cdot x \cdot x$, from R_I also applies. The other direction of the equivalence follows from the fact that $R \cup \mathrm{PROP} \cup R_I \subseteq R_\alpha$. ◀

▶ **Lemma 35.** *If $l \to r \in R_I \cup R_{II}$ and $p \neq \varepsilon$ is such that $l_{|p}$ is not a variable, then $\sigma l_{|p}$ is in R_α-normal form for any substitution σ.*

Proof. We show that $\sigma l_{|p}$ is in $R \cup \mathrm{PROP} \cup R_I$-normal form. If $l \to r \in R_I$ then $l = A_{\mathcal{T}_i} \cdot t_i$ where t_i is an R-standard term. Hence t_i is in $R \cup \mathrm{PROP} \cup R_I$-normal form. Because $p \neq \varepsilon$ and t_i is closed, this implies that $\sigma l_{|p} = l_{|p}$ is in $R \cup \mathrm{PROP} \cup R_I$-normal form as well. Analogously,

if $l \to r \in R_{II}$ then the fact that $\sigma l_{|p} = l_{|p}$ is a closed term in $R \cup \mathrm{PROP} \cup R_I$-normal form follows directly from Definition 31 and from $p \neq \varepsilon$.

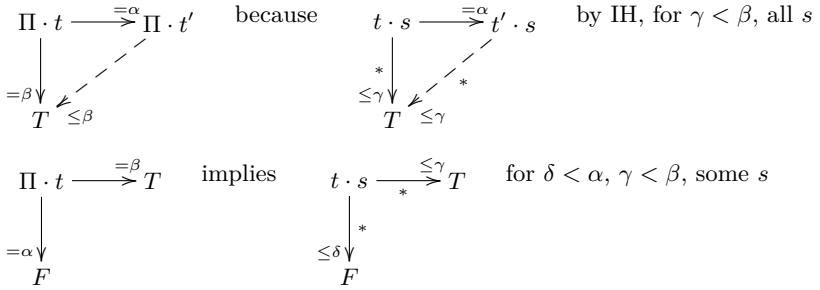
Therefore, an application of Lemma 34 establishes our claim. ◀

▶ **Lemma 36.** *If t is in $R \cup PROP$-normal form and $\Sigma_f(t) \subseteq \Sigma_f(R \cup PROP)$, then t is in R_α-normal form.*

Proof. From $\mathcal{H}(R_I \cup R_{II}) \cap \Sigma_f(R \cup \mathrm{PROP}) = \emptyset$ and Fact 18 it follows that t is in R_0-normal form. Lemma 34 implies that it is also in R_α-normal form. ◀

▶ **Notation 37.** We use S_α for $R_\alpha \setminus \bigcup_{\beta<\alpha} R_\beta$, $\to_{\leq\alpha}$ for $\to_{R_\alpha}$, and $\to_{=\alpha}$ for $\to_{S_\alpha}$.

We will now prove a series of lemmas which together imply that R_α and R_β commute for all $\alpha, \beta \leq \zeta$, and therefore R_ζ is confluent. The key idea in the proofs of these lemmas could be summarized by the following two diagrams.



We adopt the notation $\to_{*\alpha}$ for $\to_{R_\alpha \setminus R_0}$ when $\alpha > 0$, and $\to_{*0}$ for $\to_{=0}$. In the following, whenever we write a reduction sequence of the form $t_0 \overset{*}{\to}_{*\alpha_1} t_1 \overset{*}{\to}_{*\alpha_2} \ldots \overset{*}{\to}_{*\alpha_n} t_n$ we tacitly assume that there exists $\alpha \leq \zeta$ such that each α_i for $i = 1, \ldots, n$ is either 0 or α, and there is no $j \in \{1, \ldots, n-1\}$ such that $\alpha_j = \alpha_{j+1}$. It is easy to see that every reduction $t \overset{*}{\to}_{\leq\alpha} s$ can be represented by a reduction sequence in this form.

▶ **Lemma 38.** *If R_0 and $R_\alpha \setminus R_0$ commute, then so do R_0 and R_α.*

Proof. Let $t \overset{*}{\to}_{=0} t_1$ and $t = s_0 \overset{*}{\to}_{*\alpha_1} s_1 \overset{*}{\to}_{*\alpha_2} \ldots \overset{*}{\to}_{*\alpha_n} s_n = t_2$ for some $n \geq 1$ and $\alpha_1, \ldots, \alpha_n \in \{0, \alpha\}$. The proof proceeds by simple induction on n. ◀

▶ **Lemma 39.** *For all $\alpha \leq \zeta$ the term rewriting systems R_0 and $R_\alpha \setminus R_0$ commute.*

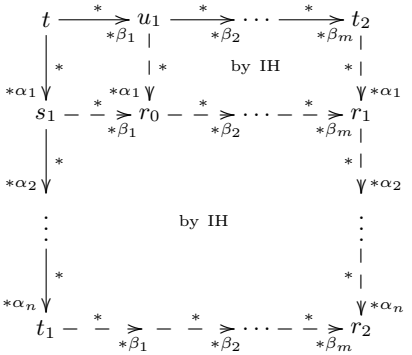
Proof. We use transfinite induction on α to show that R_0 is compatible with $R_\alpha \setminus R_0 = \bigcup_{0<\beta\leq\alpha} S_\beta$.

Let $\langle u_1, u_2 \rangle \in \mathrm{Crit}(R_0, S_\beta)$ for some $0 < \beta \leq \alpha$. Because $R \cup \mathrm{PROP}$ is standard and $\Pi \notin \Sigma_f(R \cup \mathrm{PROP})$, the critical pair must be between a rule from $R_I \cup R_{II}$ and a rule from S_β. Therefore, we have rules $l_1 \to r_1 \in R_I \cup R_{II}$, $l_2 \to r_2 \in S_\beta$, a substitution σ, and a position p such that $u_1 = \sigma r_1$, $u_2 = (\sigma l_1)[\sigma r_2]_p$, $\sigma l_{1|p} = \sigma l_2$, and p is such that $l_{1|p}$ is not a variable. If $p = \varepsilon$ then $l_1 = r_1$ by definition of R_I and R_{II}, and we have $u_1 \to_{\leq\beta} u_2$. The case $p \neq \varepsilon$ is impossible by Lemma 35.

Now let $\langle u_2, u_1 \rangle \in \mathrm{Crit}_i(S_\beta, R_0)$ for some $0 < \beta \leq \alpha$. We have $u_2 \in \{T, F\}$. There are terms t, t' and a context C such that $u_1 = \Pi \cdot C[t']$, $t \to_{=0} t'$ and $\Pi \cdot C[t] \to^{\varepsilon}_{=\beta} u_2$. Assume $u_2 = T$. The proof for $u_2 = F$ is analogous. So let s be a term in R_0-normal form. We have $C[t] \cdot s \overset{*}{\to}_{\leq\gamma} T$ for some $\gamma < \beta$. But we may invoke the inductive hypothesis to conclude that R_0 and $R_\gamma \setminus R_0$ commute. So by Lemma 38 we obtain that R_0 and R_γ commute as well. Hence $C[t'] \cdot s \overset{*}{\to}_{\leq\gamma} T$, because $C[t] \cdot s \overset{*}{\to}_{\leq\gamma} T$, $C[t] \cdot s \to_{=0} C[t'] \cdot s$ and T is in R_0-normal form. But s was an arbitrary term in R_0-normal form, so we obtain $u_1 = \Pi \cdot C[t'] \to_{=\delta} T$ for some $0 < \delta \leq \beta$. ◀

▶ **Lemma 40.** *If $R_\alpha \setminus R_0$ and $R_\beta \setminus R_0$ commute then so do R_α and R_β.*

Proof. The proof may be easily reconstructed from the following diagram. ◀



▶ **Lemma 41.** *For $\alpha', \beta' \leq \zeta$ the term rewriting systems $R_{\alpha'} \setminus R_0$ and $R_{\beta'} \setminus R_0$ commute.*

Proof. We use induction on pairs $\langle \alpha', \beta' \rangle$ of indices of $R_{\alpha'}$, $R_{\beta'}$ ordered lexicographically.

Let $\langle u_2, u_1 \rangle \in \mathrm{Crit}_i(S_\beta, S_\alpha)$ for some $0 < \alpha \leq \alpha'$, $0 < \beta \leq \beta'$. The term u_2 is a constant. There are terms t, t' and a context C such that $t \to^{\varepsilon}_{=\alpha} t'$, $u_1 = \Pi \cdot C[t']$, and $\Pi \cdot C[t] \to^{\varepsilon}_{=\beta} u_2$. Assume $u_2 = F$. There is a term s in R_0-normal form such that $C[t] \cdot s \overset{*}{\to}_{\leq\gamma} F$ for some $\gamma < \beta$. By the inductive hypothesis $R_\alpha \setminus R_0$ and $R_\gamma \setminus R_0$ commute. Therefore, by Lemma 40 we may conclude that $\to_{\leq\alpha}$ and $\to_{\leq\gamma}$ commute. Hence $C[t'] \cdot s \overset{*}{\to}_{\leq\gamma} F$, because $C[t] \cdot s \overset{*}{\to}_{\leq\gamma} F$, $C[t] \cdot s \to_{=\alpha} C[t'] \cdot s$ and F is in R_α-normal form. Therefore, $u_1 = \Pi \cdot C[t'] \to_{\leq\beta} F = u_2$. The argument for $u_2 = T$ is analogous.

Now let $\langle u_1, u_2 \rangle \in \mathrm{Crit}(S_\alpha, S_\beta)$ for some $\alpha \leq \alpha'$, $\beta \leq \beta'$. The case when $\langle u_1, u_2 \rangle$ is a non-root critical pair is analogous to the case we have just considered. If $\langle u_1, u_2 \rangle$ is a root critical pair, then both $u_1, u_2 \in \{T, F\}$, and we need to show that $u_1 = u_2$. It may happen otherwise only when there is a term t such that $\Pi \cdot t \to_{=\alpha} u_1$, $\Pi \cdot t \to_{=\beta} u_2$. Without loss of generality assume $u_1 = T$, $u_2 = F$. So there is a closed term s in R_0-normal form such that $t \cdot s \overset{*}{\to}_{\leq\delta} T$ and $t \cdot s \overset{*}{\to}_{\leq\gamma} F$ for some $\delta < \alpha$, $\gamma < \beta$. The inductive hypothesis and Lemma 40 imply that $\to_{\leq\delta}$ and $\to_{\leq\gamma}$ commute, which gives a contradiction.

We have thus shown that $R_{\alpha'} \setminus R_0$ and $R_{\beta'} \setminus R_0$ are compatible, so they commute by left-linearity and Theorem 25. ◀

▶ **Corollary 42.** *The term rewriting system R_ζ has the Church-Rosser property.*

▶ **Theorem 43.** *Let R be a standard FTRS compatible with PROP, and $\mathcal{T}_\mathbb{I} = \{\mathcal{T}_i \mid i \in \mathbb{T}\}$ be a family of sets of R-standard terms. The applicative algebra $\mathcal{A}_{R_\zeta}$ generated by $R_\zeta^{\mathcal{T}_\mathbb{I}}$ is a FO-ICA such that for each $i \in \mathbb{I}$ the set $\{[t]_{R_\zeta} \mid t \in \mathcal{T}_i\}$ is a type represented by $[A_{\mathcal{T}_i}]_{R_\zeta}$ which is a δ-total combinator. Furthermore, if $t_1, t_2 \in \mathcal{L}(\Sigma)$ are in $R \cup PROP$-normal form, $t_1 \neq t_2$, and $\Sigma_f(t_1), \Sigma_f(t_2) \subseteq \Sigma_f(R \cup PROP)$, then $[t_1]_{R_\zeta}, [t_2]_{R_\zeta} \in \delta(\mathcal{A}_{R_\zeta})$ and $[t_1]_{R_\zeta} \neq [t_2]_{R_\zeta}$.*

Proof. First, we check that $\mathcal{A}_{R_\zeta}$ is a first-order illative combinatory algebra. To save on notation we use the same symbols for terms and corresponding abstraction classes in $\mathcal{A}_{R_\zeta}$.

- The axioms $T \neq F$ and $T, F \in \delta(A_{R_\zeta})$ follow from the Church-Rosser property of R_ζ and the fact that T and F are in R_ζ-normal form.
- The axioms (3)-(6) in Definition 2 follow directly from the definition of PROP.
- The axioms (7) and (8) follow directly from the definitions of R_I, R_{II} and from Lemma 34, which is needed to prove that $A \cdot X \in \{F\} \cup \upsilon$ for $X \in \upsilon$.

- The axioms for Π follow from the Church-Rosser property of R_ζ, Lemma 34 and the fact that $R_\zeta = \bigcup_{\alpha<\zeta} R_\alpha$.

The fact that each set $\{[t]_{R_\zeta} \mid t \in \mathcal{T}_i\}$ is a type represented by $[A_{\mathcal{T}_i}]_{R_\zeta}$ which is a δ-total combinator follows directly from Lemma 34 and the definitions of R_I and R_{II}. Finally, the last claim follows from Lemma 36 and the Church-Rosser property of R_ζ. ◀

6 Completeness of the first-order translation

In this section we prove completeness of the translation introduced in Section 3. We work under the same assumptions and definitions as in Section 3.

▶ **Theorem 44. Completeness**
Let ϕ and all formulas in Δ be closed. If $\Psi(\Delta) \models \Psi(\phi)$ then $\Delta \models_{FO} \phi$.

Proof. Suppose $\Psi(\Delta) \models \Psi(\phi)$. Let $\mathcal{A}$ be a first-order model of Δ.

We construct a functional term rewriting system R as follows. The signature of R consists of all elements of the universe of $\mathcal{A}$, all relation and function symbols from $\mathcal{L}_{FO}$ and the constants T, F. We assume the relation and function symbols are different from T, F, P, Q, etc. For every n-ary relation $r^{\mathcal{A}}$ on $\mathcal{A}$, which interprets a relation symbol r, the rule $r \cdot a_1 \cdot \ldots \cdot a_n \to T$ belongs to R for exactly those $a_1, \ldots, a_n$ for which $r^{\mathcal{A}}(a_1, \ldots, a_n)$ holds, the rule $r \cdot a_1 \cdot \ldots \cdot a_n \to F$ when $r^{\mathcal{A}}(a_1, \ldots, a_n)$ does not hold. For every n-ary function $f^{\mathcal{A}}$ on $\mathcal{A}$, which interprets a function symbol f, the rule $f \cdot a_1 \cdot \ldots \cdot a_n \to b$ belongs to R if $f^{\mathcal{A}}(a_1, \ldots, a_n) = b$. Nothing else belongs to R.

It is straightforward to verify that R is standard and compatible with PROP. By ς we denote the universe of $\mathcal{A}$. Let $\mathcal{B}$ be the applicative algebra generated by $R_\zeta^{\{\varsigma\}}$. For convenience we use the same symbols for terms and corresponding abstraction classes. Analogously for sets of terms. By Theorem 43 the algebra $\mathcal{B}$ is a FO-ICA with a δ-total combinator A_ς representing ς, and we have $\varsigma \subseteq \delta(\mathcal{B})$. Note that $\varsigma = \iota_{\mathcal{B}}$, where $\iota_{\mathcal{B}}$ is the type represented by $A_\iota = \lambda x.(A_\varsigma x) \wedge (A_\delta x)$, as in Definition 10.

It is easy to check that $\mathcal{B}$ is an illative model of Γ_0, and that $\mathcal{A}$ and $\mathcal{B}$ are correspondent in the sense of Definition 11. Hence by Lemma 12 we may conclude that $[\![\psi]\!]_{\mathcal{A}} = [\![\Psi(\psi)]\!]_{\mathcal{B}}$ for any closed first-order formula ψ. This implies that $\mathcal{B} \models \mathrm{Im}_\Psi(\Delta)$. Therefore $\mathcal{B} \models \Psi(\Delta)$, and consequently $\mathcal{B} \models \Psi(\phi)$, which implies $\mathcal{A} \models_{FO} \phi$, because $[\![\phi]\!]_{\mathcal{A}} = [\![\Psi(\phi)]\!]_{\mathcal{B}}$. ◀

References

1. Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1999.
2. Henk Barendregt, Martin W. Bunder, and Wil Dekkers. Systems of illative combinatory logic complete for first-order propositional and predicate calculus. *Journal of Symbolic Logic*, 58(3):769–788, 1993.
3. Alonzo Church. A set of postulates for the foundation of logic I. *Annals of Mathematics, ser. 2*, 33:346–366, 1932.
4. Haskell B. Curry. Grundlagen der kombinatorischen Logik. *American Journal of Mathematics*, 52:509–536, 789–834, 1930.
5. Wil Dekkers, Martin W. Bunder, and Henk Barendregt. Completeness of the propositions-as-types interpretation of intuitionistic logic into illative combinatory logic. *Journal of Symbolic Logic*, 63(3):869–890, 1998.

6 Wil Dekkers, Martin W. Bunder, and Henk Barendregt. Completeness of two systems of illative combinatory logic for first-order propositional and predicate calculus. *Archive for Mathematical Logic*, 37(5-6):327–341, 1998.

7 Gerhard Jäger, Reinhard Kahle, and Thomas Strahm. On applicative theories. In A. Cantini, E. Casari, and P. Minari, editors, *Logic and Foundation of Mathematics*, pages 88–92. Kluwer Academic Publishers, 1999.

8 Gerhard Jäger and Thomas Strahm. Totality in applicative theories. *Annals of Pure and Applied Logic*, 74(2):105–120, 1995.

9 Stephen C. Kleene and J. Barkley Rosser. The inconsistency of certain formal logics. *Annals of Mathematics*, 36:630–636, 1935.

10 Saul A. Kripke. Outline of a theory of truth. *Journal of Philosophy*, 72(19):690–716, 1975.

11 Jonathan P. Seldin. The logic of Church and Curry. In Dov M. Gabbay and John Woods, editors, *Logic from Russell to Church*, volume 5 of *Handbook of the History of Logic*, pages 819–873. North-Holland, 2009.

12 Yoshihito Toyama. Commutativity of term rewriting systems. In K. Fuchi and L. Kott, editors, *Programming of Future Generation Computers II*, pages 393–407. North-Holland, 1988.

Enumeration Complexity of logical query problems with second order variables

Arnaud Durand and Yann Strozecki

Université Paris Diderot, IMJ, Projet Logique, CNRS UMR 7586
Case 7012, 75205 Paris cedex 13, France
durand@logique.jussieu.fr, strozecki@logique.jussieu.fr

Abstract

We consider query problems defined by first order formulas of the form $\Phi(\mathbf{x}, \mathbf{T})$ with free first order and second order variables and study the data complexity of enumerating results of such queries. By considering the number of alternations in the quantifier prefixes of formulas, we show that such query problems either admit a constant delay or a polynomial delay enumeration algorithm or are hard to enumerate. We also exhibit syntactically defined fragments inside the hard cases that still admit good enumeration algorithms and discuss the case of some restricted classes of database structures as inputs.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Descriptive complexity, enumeration, query problem

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.189

Introduction

Query answering for logical formalisms is a fundamental problem in database theory. There are two natural ways to consider the answering process and consequently to evaluate the complexity of such problems. Given a query φ on a database structure $\mathcal{S}$, one may consider computing the result $\varphi(\mathcal{S})$ as a global process and measure its data complexity in terms of the database and the output sizes. Alternatively, one can see this task as a dynamical process in which one computes the tuples of the solution set one after the other. In this case, the main measure is the delay spent between two successive output tuples. In recent years, this approach has deserved some attention in the context of logical query problems: see, for example, [3] for a study on conjunctive queries, or [4, 1] for monadic second order logic on bounded tree-width structures or [6, 11] for first order queries on structures of bounded degree. However, having only free first order variables in formulas is not enough to capture complex objects of non-constant size. This is the case when one wants to obtain, for example, cliques or hypergraph transversals of arbitrary size (see Example 2) or classical NP properties.

It is known since Fagin's theorem [7] that NP corresponds exactly to problems definable in existential second order logic. That is, the language L is in NP, if and only if there exists an existential second order formula $\Phi(\mathbf{T})$ over a signature $\sigma \cup \{\mathbf{T}\}$, such that, for all σ-structure $\mathcal{S}$:

$$\mathcal{S} \in L \iff \mathcal{S} \models \exists \mathbf{T}\ \Phi(\mathbf{T}).$$

In this paper, we consider first order query with possibly free second order variables and study their enumeration complexity. Since in full generality such formulas may be very

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 189–202

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

expressive, we consider fragments defined by the quantifier alternation of formulas[1]. The hardness of counting the number of solutions of a problem may sometimes be seen as a first approach of enumeration complexity. In [12], a descriptive complexity point of view of counting problems is proposed. They show that for the $\mathbf{\Sigma}_0$ (quantifier free) fragment, counting the number of solutions can be done in polynomial time and that already at the first universal level $\mathbf{\Pi}_1$ (only one block of universal quantifiers) ♯P-complete problems can be defined. They also show that the $\mathbf{\Sigma}_1$ level (one block of existential quantifiers) and another syntactically defined fragment admit a fully polynomial randomized approximation scheme to count the number of solutions. In this paper, we show that the situation for enumeration is more complex. Our contributions are as follows.

- For any fixed formula $\Phi(\mathbf{x}, \mathbf{T}) \in \mathbf{\Sigma}_0$, there exists an algorithm that, given a structure $\mathcal{S}$, enumerates $\Phi(\mathcal{S})$ with polynomial time precomputation and constant delay. Under a parameterized complexity assumption, the degree of the polynomial in the precomputation step depends on the formula size. To show constant delay enumeration we prove that one can pass from one solution (of the form $(\mathbf{x}, \mathbf{T})$) to another by only a constant number of local changes.
- We also prove that, for any k, if the structure $\mathcal{S}$ is of degree bounded by k, then there is an enumeration algorithm for the $\mathbf{\Sigma}_0$ fragment with linear precomputation and constant delay.
- For any fixed formula $\Phi(\mathbf{x}, \mathbf{T}) \in \mathbf{\Sigma}_1$, there exists an algorithm that, given a structure $\mathcal{S}$, enumerates $\Phi(\mathcal{S})$ with polynomial time precomputation and polynomial time delay. To this aim, we study the closure under union problem in the context of enumeration and prove that some closure result holds even for the union of two problems which are efficiently enumerable but relatively to different orderings (of their respective solution space).
- The class $\mathbf{\Pi}_1$ already contains problems that are hard to enumerate and $\mathbf{\Pi}_2$ is enough to capture all **FO** definable problems on ordered structures up to parsimonious reductions.
- Finally, we exhibit natural fragments above $\mathbf{\Pi}_1$ that admit efficient enumeration procedures.

Basic definitions about logical query problems, enumeration problems and main enumeration complexity measures are given in Section 1. Results about the enumeration complexity of $\mathbf{\Sigma}_0$ query problems are given in Section 2 and about the $\mathbf{\Sigma}_1$ query problems in Section 3. In this latter section, we also discuss the intimate relationship between the enumeration of models of propositional formulas in disjunctive normal form and $\mathbf{\Sigma}_1$ query problems. The $\mathbf{\Pi}_1$ fragment is studied in Section 4 where both the hardness results are given and some tractable fragments are exhibited.

1 Preliminaries

Enumeration problem and complexity

Let $\mathcal{I}, \mathcal{O}$ be two sets and R be a polynomially balanced binary predicate $R \subseteq \mathcal{I} \times \mathcal{O}$ decidable in polynomial time. In particular, given $x \in \mathcal{I}$ and $y \in \mathcal{O}$, checking whether $R(x, y)$ can be done in time polynomial in $|x|$. One defines the enumeration function associated to R as follows.

[1] Note that this is the approach to define the classes of the **W** hierarchy in parameterized complexity

ENUM·R
Input: $x \in \mathcal{I}$
Output: an enumeration of elements in $R(x) = \{y : R(x, y)\}$

In this paper, we consider the *random access machine* model (RAM) with addition and subtraction as its basic arithmetic operations. It has read-only *input registers* $I_1, I_2, \ldots$ (containing the input x), read-write *work registers* $R_1, R_2, \ldots$ and *output registers* $O_1, O_2, \ldots$. Our model is equipped with an additional instruction Output which, when executed, indicates that the non empty output registers contain a partial output $y \in R(x)$. Time complexity is used under the *uniform cost* model. A RAM is of space complexity $O(h(n))$ if, for all inputs of size n, it uses working registers R_i of addresses $i = O(h(n))$ and content $O(\max(n, h(n)))$.

A scheme $\mathcal{A} = (\mathcal{A}_p, \mathcal{A}_e)$ (see [2] for a similar definition) computes the enumeration problem ENUM·R if, for any input x:

- $\mathcal{A}_p$ computes from x an extended input $ext(x)$. This is called the precomputation phase.
- Given $ext(x)$, $\mathcal{A}_e$ computes one after the other and without repetition the elements of $R(x)$ and stops immediately after writing the last one.

We denote by $\text{time}_j(x)$ the moment when $\mathcal{A}$ has completed the writing of the j^{th} solution i.e. after the j^{th} Output instruction is executed (by convention, $\text{time}_0(x) = 0$). Let $\text{delay}_j(x) = \text{time}_j(x) - \text{time}_{j-1}(x)$.

▶ **Definition 1.** Let $g : \mathbb{N} \to \mathbb{N}$, $f : \mathbb{N} \to \mathbb{N}$ be two functions. The problem ENUM·R belongs to the class DELAY(g, f) if there exists an enumeration scheme $\mathcal{A} = (\mathcal{A}_p, \mathcal{A}_e)$ that computes ENUM·R such that, for all input x:

- Precomputation uses time and space $O(g(|x|))$,
- Solutions $y \in R(x)$ are computed successively from $ext(x)$ using delay $O(f(|x|))$ and space $O(\max_{y \in R(x)}(f(|x|), |y|))$

The two enumeration classes below are classical (see the second chapter of [13] and the references therein):

$$\text{DELAYP} = \bigcup_{k,h} \text{DELAY}(n^k, n^h), \quad \text{CONSTANT-DELAY} = \bigcup_{k} \text{DELAY}(n^k, 1).$$

Logical definitions

We suppose the reader is familiar with the basics of finite model theory and first order logic [10]. A signature $\sigma = \{R_1, ..., R_k\}$ is a set of relational symbols (constant symbols will also be authorized). The arity of a predicate R_i is denoted by $\text{ar}(R_i)$. A σ-structure $\mathcal{S} = \langle D, R_1^{\mathcal{S}}, \ldots, R_k^{\mathcal{S}} \rangle$ is composed of a domain D, together with an interpretation $R_i^{\mathcal{S}} \subseteq D^{\text{ar}(R_i)}$ for symbols R_i of σ. When the context is clear, the interpretation $R_i^{\mathcal{S}}$ of R_i is denoted by R_i^*. The size of $\mathcal{S}$ is equal to the cardinality $|D|$ of its domain plus the sum of the number of tuples times the arity for all relations. It is denoted by $|\mathcal{S}|$. If $n \in \mathbb{N}$ such that $|D| = n$ then, D will often be identified with the initial segment of the positive integers $[n]$.

Let σ be a signature and $\mathbf{T} = (T_1, \ldots, T_h)$ be a tuple of predicate symbols not in σ, let $\mathbf{z} = (z_1, \ldots, z_l)$ be a tuple of variables. We consider first order formulas $\Phi(\mathbf{z}, \mathbf{T})$ with free first order and second order variables. Such formulas, of signature $\sigma \cup \mathbf{T}$ have atomic formulas (atoms) built over relations of $\sigma \cup \mathbf{T}$ and equality symbol $=$. We denote by $\mathbf{\Sigma}_0$ (or $\mathbf{\Pi}_0$) the set of quantifier free first order formulas. A formula $\Phi(\mathbf{z}, \mathbf{T})$ is in $\mathbf{\Sigma}_{i+1}$ (resp. $\mathbf{\Pi}_{i+1}$), for $i \geq 0$, if it is of the form: $\exists \mathbf{x} \psi$ (resp. $\forall \mathbf{x} \psi$) where ψ is in $\mathbf{\Pi}_i$ (resp. $\mathbf{\Sigma}_i$).

Enumeration Query problems and data complexity

Let $\mathscr{F}$ be a subclass of first order formulas and $\Phi(\mathbf{z},\mathbf{T}) \in \mathscr{F}$, we consider the following variant of the classical query problem.

ENUM·Φ
Input: A σ-structure $\mathcal{S}$
Output: an enumeration of elements in $\Phi(\mathcal{S}) = \{(\mathbf{z}^*,\mathbf{T}^*) : (\mathcal{S},\mathbf{z}^*,\mathbf{T}^*) \models \Phi(\mathbf{z},\mathbf{T})\}$

We denote by ENUM·$\mathscr{F}$ the collection of problems ENUM·Φ for $\Phi \in \mathscr{F}$. Note that it can be supposed without loss of generality that the tuple $\mathbf{T}$ contains only one relation T of arity $r = \max_{i\leq h} \mathrm{ar}(T_i) + 1$. To do this, one simply represents each predicate T_i by T and a new constant symbol a_i and replace in formulas each $T_i(\mathbf{x})$ by $T(\mathbf{a}_i,\mathbf{x})$ where the length of $\mathbf{a}_i$ is $r - \mathrm{ar}(T_i)$. It suffices to add the new constants in the signature.

▶ **Example 2.** The formula $IS(T) \equiv \forall x \forall y\ T(x) \wedge T(y) \Rightarrow \neg E(x,y)$ holds if and only if T is an independent set. Remark that the previous formula is in $\mathbf{\Pi}_1$, thus ENUM·IS is in ENUM·Π_1.

▶ **Example 3.** ENUM·HS : given a hypergraph H, enumerate the hitting sets (vertex covers) of H. The hypergraph H is represented by the incidence structure $\langle D, \{V,E,R\}\rangle$ where $V(x)$ means that x is a vertex, $E(y)$ that y is an hyperedge and $R(x,y)$ that x is a vertex of the hyperedge y.

$$HS(T) \equiv \forall x(T(x) \Rightarrow V(x)) \wedge \forall y \exists x E(y) \Rightarrow (T(x) \wedge R(x,y))$$

Therefore the problem ENUM·HS is in ENUM·$\mathbf{\Pi}_2$.

Note that in the query problem ENUM·Φ the formula is fixed i.e. is not part of the input. The complexity is evaluated in terms of the structure/data only. For such problems, the notion of constant delay makes sense:
- when the free variables are all first order (in that case each output is of constant size)
- *but also* and more interestingly when there are second order variables and that computing the next solution from the preceding one can be done by changing a constant number of tuples.

2 Enumeration for $\mathbf{\Sigma}_0$ formulas

In this section, we give enumeration algorithms for the most simple class of ENUM·$\mathbf{\Sigma}_0$. Since it is a core procedure of our algorithms, we need to recall how to enumerate all k-ary relations over any domain with constant delay.

▶ **Lemma 4** (Gray code enumeration). *Let D be a finite set, $k \in \mathbb{N}$ and $t_1,\ldots,t_a$, $s_1,\ldots,s_b$ in D^k. Let $\mathcal{R} = \{R \subseteq D^k : t_1,\ldots,t_a \in R, s_1,\ldots,s_b \notin R\}$. Then, starting from the relation $R = \{t_1,\ldots,t_a\}$, one can enumerate the relations belonging to $\mathcal{R}$ with precomputation and delay in $O(1)$. Moreover, the process ends by producing a relation R' such that $|R'| = |R|+1$.*

Proof. Since the tuples $t_1,...,t_a$ must belong to each output, they can simply be fixed and the problem reduces to generate all subsets of $D^k \backslash \{t_1,\ldots,t_a,s_1,\ldots,s_b\}$ of size up to $n = |D|^k - a - b$ starting from the empty set. Clearly, it is equivalent to generate all subsets of $[n]$. Such problems have been widely studied under the name of Gray code enumeration. It is well known that the enumeration can be done in such a way that the size of the symmetric difference $R_1 \Delta R_2$ between two successive outputs R_1 and R_2 is 1. Given an output R_1, one

can proceed as follows. If R_1 has an even number of elements, then set $R_2 = R_1\Delta\{n\}$. If not, let $R_2 = R_1\Delta\{i-1\}$ where i is the greatest element in R_1. Clearly, the delay is constant provided we have access to the information on the parity of number of elements in R_1 and on the value of such i above. The parity can be stored in one bit, that is changed at each step, while the latter is easy to maintain in constant time by a linked list on the tuples of each produced relation. To start, one only needs to build this data structure on the first relation R which is of size a. It is easy to see that the enumeration ends up with the relation R' containing $t_1, ..., t_a$ and the tuple of $D^k\backslash\{t_1, \ldots, s_b\}$ indexed by 1. ◀

▶ Remark. Note that the memory space required by the preceding algorithm is linear in n, the size of one output. It may seem important since, in contrast, the enumeration itself is constant delay. However, some data structure is required only to navigate inside each output relation and make the necessary local changes easily.

There is a standard way to represent a first order query problem by a propositional satisfiability problem. We recall it below. We will later introduce a more complex representation.

Let σ be a relational signature and let $\mathcal{S}$ be a σ-structure of domain D with $|D| = n$. Let $\Phi(\mathbf{z}, T)$ be a first order formula where $\mathbf{z}$ is a k-tuple of first order variables and T is a second order variable of arity r. One rewrites $\Phi(\mathbf{z}, T)$ by $\bigvee_{i=0}^{n^k-1} \Phi(\mathbf{z}_i, T)$ where $\mathbf{z}_i$ is the i^{th} element of D^k (for, say, lexicographic ordering on D^k). In each $\Phi(\mathbf{z}_i, T)$ one replaces inductively (bottom-up in the tree representation of the formula) each sub-formula $\exists\mathbf{y}\varphi(\mathbf{z}_i, \mathbf{y}, T)$ by a disjunction $\bigvee_{j=0}^{n^p-1} \varphi(\mathbf{z}_i, \mathbf{y}_j, T)$ with $|\mathbf{y}| = p$ (and similarly universal quantification by conjunction). Finally, one calls $\tilde{\Phi}_i$ the propositional formula obtained from $\Phi(\mathbf{z}_i, T)$ by replacing every atomic formula $R(\mathbf{w})$ with $R \in \sigma$ by its truth value in $\mathcal{S}$ and we set $\tilde{\Phi} = \bigvee_{i=0}^{n^k-1} \tilde{\Phi}_i$. Variables of $\tilde{\Phi}$ are of the form $T(\mathbf{w})$ with $\mathbf{w} \in D^r$.

We are now ready to state the first result of this section.

▶ **Theorem 5.** Enum·$\mathbf{\Sigma}_0 \subseteq$ Constant-Delay. *More precisely, it can be computed with precomputation $O(|D|^k)$ and delay $O(1)$ where k is the number of free first order variables of the formula and D is the domain of the input structure.*

Proof. Let $\mathcal{S}$ be a σ-structure of domain D. Let $\Phi(\mathbf{z}, T) \in \mathbf{\Sigma}_0$ with T of arity r and let $\tilde{\Phi} = \bigvee_{i=0}^{n^k-1} \tilde{\Phi}_i$ be its associated propositional formula.

The idea of the proof consists in determining some canonical assignments for each $\tilde{\Phi}_i$ from which one can enumerate all models of $\tilde{\Phi}_i$ and then all models of $\tilde{\Phi}$ by disjoint union. Since constant delay is expected, one has to be careful that two consecutive partial enumerations, say for models of $\tilde{\Phi}_i$ and of $\tilde{\Phi}_j$ with $i \neq j$, respectively ends and starts with solutions that are "close" to each other.

Since there is no first order variable other than $\mathbf{z}$, the number of propositional variables appearing in each $\tilde{\Phi}_i$ is bounded by a constant c_i independent of $|\mathcal{S}|$. Let $T(\mathbf{y}_{i,j})$, $j \leq c_i$, be such variables with $\mathbf{y}_{i,j} \in D^r$.

Let $\mathcal{I}(\Phi_i)$ be the set of up to 2^{c_i} models of $\tilde{\Phi}_i$. For $I \in \mathcal{I}(\Phi_i)$, let I_0 (resp. I_1) the set of variables set to false (resp. true) in I. Let $T(\mathbf{z}_i, I)$ be the set of r-uples $\mathbf{y}_{i,j}$ such that $T(\mathbf{y}_{i,j}) \in I_1$. This relation contains at most c_i tuples. Let now $[T(\mathbf{z}_i, I)]$ be the set of relations generated by $T(\mathbf{z}_i, I)$ i.e. the relations T^* that agrees with $T(\mathbf{z}_i, I)$ on $\mathbf{y}_{i,1}, \ldots, \mathbf{y}_{i,c_i}$. Clearly, for each $i \leq n^k - 1$, the set $\Phi(\mathcal{S})\}$ is equal to the set:

$$\bigcup_{I\in\mathcal{I}(\Phi_i)} \bigcup_{T^*\in[T(\mathbf{z}_i,I)]} (\mathbf{z}_i, T^*).$$

The enumeration process to compute $\Phi(\mathcal{S})$ when $\Phi \in \mathbf{\Sigma}_0$ can now be described. The precomputation steps are as follows.

- For each $\mathbf{z}_i \in D^k$, compute $\tilde{\Phi}_i$ and the set $\mathcal{I}(\Phi_i)$.
- Compute the set $\mathcal{Z} = \{\mathbf{z}_i \in D^k : \mathcal{I}(\Phi_i) \neq \emptyset\}$.

It holds that $|\mathcal{I}(\Phi_i)| \leq 2^{c_i}$ i.e. is constant. Thus, the precomputation requires time $O(|D|^k)$. Now, the enumeration itself proceeds as follows.

- For each $\mathbf{z}_i \in \mathcal{Z}$, for each $I \in \mathcal{I}(\Phi_i)$, generate all relations $T^* \in [T(\mathbf{z}_i, I)]$ and output $(\mathbf{z}_i, T^*)$.

From Lemma 4, for given $\mathbf{z}_i$ and I, one can enumerate the set $[T(\mathbf{z}_i, I)]$ in delay $O(1)$. Note that for two distinct $\mathbf{z}_i$ and $\mathbf{z}_j$, the set of outputs are disjoints. Similarly, since two distinct assignments $I, I' \in \mathcal{I}(\Phi_i)$ differ for at least one variable, it holds that $[T(\mathbf{z}_i, I)] \cap [T(\mathbf{z}_i, I')] = \emptyset$.

For each $\mathbf{z}_i$ and I, one starts the enumeration with the relation $T(\mathbf{z}_i, I)$ of size less than c_i and ends by a relation $T'(\mathbf{z}_i, I) \in [T(\mathbf{z}_i, I)]$ with $|T'(\mathbf{z}_i, I)| = |T(\mathbf{z}_i, I)| + 1$. Then, for all $I' \in \mathcal{I}(\Phi_i)$, $I' \neq I$:

$$|T'(\mathbf{z}_i, I) \Delta T(\mathbf{z}_i, I')| \leq 2c_i + 1.$$

Similarly, for all $\mathbf{z}_j \neq \mathbf{z}_i$, and all $I' \in \mathcal{I}(\Phi_j)$ it holds:

$$|T'(\mathbf{z}_i, I) \Delta T(\mathbf{z}_j, I')| \leq c_i + c_j + 1.$$

Then, the enumeration process remains constant delay when branching from one assignment I to the next and when branching from one $\mathbf{z}_i$ to the next. ◀

Is it possible to improve Theorem 5 to find a constant delay enumeration algorithm for $\mathbf{\Sigma}_0$ formulas with a *fixed* polynomial (i.e. of degree independent of the formula size) precomputation? A partial negative answer comes from the following remark. Note that the k-Clique problem can be expressed at this level on finite ordered graph. For instance, for $k = 3$ (see [12]):

$$\Phi(z_1, z_2, z_3) \equiv z_1 < z_2 \wedge z_2 < z_3 \wedge E(z_1, z_2) \wedge E(z_2, z_3) \wedge E(z_3, z_1)$$

Recall that the precomputation plus the delay (which is constant in Theorem 5) correspond to the time necessary to produce the first output, hence to decide if the problem has at least one solution. Then, a *fixed* polynomial precomputation for Enum·$\mathbf{\Sigma}_0$ would provide a fixed parameter tractable algorithm for the parameterized clique problem (see [8] for definition and references on parameterized complexity). Such an algorithm is generally not believed to exist (unless the two parameterized classes $\mathbf{W}[1]$ and $\mathbf{FPT}$ coincide). However, as shown below, such an improvement of Theorem 5 can be found for some restricted class of structures as input.

A structure $\mathcal{S} = \langle D, R_1, \ldots, R_i \rangle$ is of degree bounded by $d \in \mathbb{N}$ (i.e. is d-degree bounded), if for every $x \in D$, x occurs in at most d tuples of each relation R_i. The following result shows that in the case of bounded degree structures as input an algorithm with linear precomputation can be found. It is proved by using a representation of the query problem by a mixed problem combining querying (but without second order variable) and satisfiability testing.

▶ **Theorem 6.** *Let $d \in \mathbb{N}$. On d-degree bounded input structures,* Enum·$\mathbf{\Sigma}_0 \in$ Delay$(|D|, 1)$ *where D is the domain of the input structure $\mathcal{S}$.*

Proof. One difference with the proof of Theorem 5 is that $\Phi(\mathbf{z}, T)$ is now represented by a pair made of a propositional formula $\overline{\Phi}$ and a set of $\mathbf{\Sigma}_0$ formulas (interpreted on bounded structure) without second order free variables.

Let c be the total number of distinct atomic formulas that appears in $\Phi(\mathbf{z}, T)$. Each atom is of the form $T(\mathbf{y})$ or of the form $R(\mathbf{x})$ with $R \in \sigma$ and $\mathbf{x}, \mathbf{y}$ subsets of $\mathbf{z}$. Clearly, $\Phi(\mathbf{z}, \mathbf{T})$ can be seen as an "abstract" propositional formula denoted by $\overline{\Phi}$ over propositional variables $T(\mathbf{y})$ and $R(\mathbf{x})$ where $\mathbf{y}$ and $\mathbf{x}$ are simply viewed as indices. Let $\mathcal{J}(\overline{\Phi})$ be the set of up to 2^c models of $\overline{\Phi}$. One can recover elements $(\mathbf{z}^*, \mathbf{T}^*)$ of $\Phi(\mathcal{S})$ from the satisfying assignments of $\overline{\Phi}$ as follows. Let $J \in \mathcal{J}(\overline{\Phi})$ and J_0 (resp. J_1) the set of variables set to false (resp. true) in J. Let us consider the first-order formula $\varphi_J(\mathbf{z})$ on signature σ below:

$$\bigwedge_{T(\mathbf{y}) \in J_0} \bigwedge_{T(\mathbf{y}') \in J_1} \bigvee_{j=1}^{r} y_j \neq y'_j \wedge \bigwedge_{R(\mathbf{x}) \in J_1} R(\mathbf{x}) \wedge \bigwedge_{R(\mathbf{x}) \in J_0} \neg R(\mathbf{x}),$$

Let also:

$$\varphi_J(\mathcal{S}) = \{\mathbf{z}^* : \langle S, \mathbf{z}^* \rangle \models \varphi_J(\mathbf{z})\}.$$

For $\mathbf{z}^* \in \varphi_J(\mathcal{S})$, we denote by $J(\mathbf{z}^*)$ the truth assignments of the c-tuples (of the form $T(\mathbf{y})$ or $R(\mathbf{x})$ with $\mathbf{x}$ and $\mathbf{y}$ subsets of variables taken from $\mathbf{z}$) induced by J after instantiation of the variables in $\mathbf{z}$ by $\mathbf{z}^*$. In other words, in $J(\mathbf{z}^*)$, $T^*(\mathbf{y}^*)$ is true iff $T(\mathbf{y}) \in J_1$ and $R^*(\mathbf{x}^*)$ is true iff $R(\mathbf{x}) \in J_1$.

We now compare with the formulas $\tilde{\Phi}$ in Theorem 5. The following are true:

- Let $J \in \mathcal{J}(\overline{\Phi})$ and $\mathbf{z}_i$, the ith element of D^k. Suppose that $\mathbf{z}_i \in \varphi_J(S)$ then, $J(\mathbf{z}_i) \in \mathcal{I}(\Phi_i)$.

- Conversely, let $\mathbf{z}_i \in D^k$ and $I \in \mathcal{I}(\Phi_i)$ then, there exists $J \in \mathcal{J}(\overline{\Phi})$ such that $\mathbf{z}_i \in \varphi_J(S)$ and $I = J(\mathbf{z}_i)$

From the discussion above, the following holds:

$$\Phi(\mathcal{S}) = \bigcup_{J \in \mathcal{J}(\overline{\Phi})} \bigcup_{\mathbf{z}^* \in \varphi_J(S)} \bigcup_{T^* \in [T(z^*, J(\mathbf{z}^*))]} (\mathbf{z}^*, T^*) \tag{1}$$

Let J and J' be distinct assignments. Observe that, if there exists an atomic formula $R(\mathbf{x})$ over which J and J' has a different value then $\varphi_J(S) \cap \varphi_{J'}(S) = \emptyset$. In this case, the two sets

$$\bigcup_{\mathbf{z}^* \in \varphi_J(S)} \bigcup_{T^* \in [T(z^*, J(\mathbf{z}^*))]} (\mathbf{z}^*, T^*) \text{ and } \bigcup_{\mathbf{z}^* \in \varphi_{J'}(S)} \bigcup_{T^* \in [T(z^*, J'(\mathbf{z}^*))]} (\mathbf{z}^*, T^*)$$

are obviously disjoint. Moreover, if J and J' agree on all atomic formulas of the form $R(\mathbf{x})$, then they differ on at least one $T(\mathbf{y})$ and, in this case $[T(z^*, J(\mathbf{z}^*))] \cap [T(z^*, J'(\mathbf{z}^*))] = \emptyset$. Thus the two above sets are also disjoint even if there might exist $\mathbf{z}^* \in \varphi_J(S) \cap \varphi_{J'}(S)$.

We can now describe how to enumerate $\Phi(\mathcal{S})$. The precomputation process is as follows.

- Compute $\overline{\Phi}$, $\mathcal{J}(\overline{\Phi})$ and, for each $J \in \mathcal{J}(\overline{\Phi})$, the formula $\varphi_J(\mathbf{z})$. All this can be achieved in constant time.

- For each $J \in \mathcal{J}(\overline{\Phi})$, run the necessary precomputation phase to enumerate the elements of $\varphi_J(\mathcal{S})$. From [6] it is known that enumerating the result of a first order query over a structure of bounded degree i.e. computing $\varphi_J(\mathcal{S})$ can be done with a $O(|D|)$ precomputation and a $O(1)$ delay. Hence, the total precomputation phase requires $O(|D|)$ steps.

For the enumeration phase, we conclude as for Theorem 5, taking into account that all components in Equation (1) are pairwise disjoints. ◀

▶ Remark. Each query $\varphi_J(\mathbf{z})$ in the above proof is evaluated on a bounded degree structure which makes the global enumeration tractable. However, the representation of a $\mathbf{\Sigma}_0$ formula $\Phi(\mathbf{z}, T)$ by an abstract propositional formula $\overline{\Phi}$ and a collection of $\mathbf{\Sigma}_0$ formulas $\varphi_J(\mathbf{z})$ without second order variable is general. Then, if $\mathscr{S}$ is any class of structures on which queries of the form $\varphi_J(\mathbf{z})$ admit a linear precomputation and constant delay algorithm then, on $\mathscr{S}$, it also holds that $\textsc{Enum}\cdot\mathbf{\Sigma}_0 \subseteq \textsc{Delay}(|D|, 1)$.

3 Enumeration for Σ_1 formulas

In this section, we prove a lemma, which allows to enumerate the union of the solutions of two enumeration problems with a manageable delay. It is then used to prove that $\textsc{Enum}\cdot\mathbf{\Sigma}_1 \subseteq \textsc{DelayP}$.

▶ **Definition 7.** Let $R(x, y)$ and $S(x, y)$ be two polynomially balanced predicates. The union of R and S, denoted by $(R \cup S)$, is defined by: for all x, y, $(R \cup S)(x, y)$ holds if and only if $R(x, y)$ holds or $S(x, y)$ holds.

Recall that $R(x)$ is the finite set $\{y \mid R(x, y)\}$. Assume that $\textsc{Enum}\cdot R$ and $\textsc{Enum}\cdot S$ are in $\textsc{Delay}(g(n), f(n))$. If, for all x, $R(x) \cap S(x) = \emptyset$ then $\textsc{Enum}\cdot(R \cup S) \in \textsc{Delay}(g(n), f(n))$. Similarly, if there exist algorithms with precomputation $g(n)$ and delay $f(n)$ that enumerate the solutions of $\textsc{Enum}\cdot R$ and $\textsc{Enum}\cdot S$ with respect to the same linear ordering $<$ on the output space, then $\textsc{Enum}\cdot(R \cup S) \in \textsc{Delay}(g(n), f(n))$ ([6, 2]). The following result shows that some kind of closure under union can be established without disjointness conditions nor assumption on the ordering of enumeration.

Algorithm 1: Enumeration algorithm for $\textsc{Enum}\cdot(R \cup S)$

```
Data: An instance x
Result: The elements of R(x) ∪ S(x)
y1 ⟵ First element of the enumeration of R(x)
y2 ⟵ First element of the enumeration of S(x)
while y1 ≠ END ∨ y2 ≠ END do
    if y1 ≠ END ∧ y1 ∉ S(x) then
        Output y1
    else
        Output y2 ;
        y2 ⟵ next element of the enumeration of S(x)
if y1 ≠ END then
    y1 ⟵ next element of the enumeration of R(x)
```

▶ Proposition 8. Let $f : \mathbb{N} \to \mathbb{N}$, $g : \mathbb{N} \to \mathbb{N}$, $h : \mathbb{N} \to \mathbb{N}$ and R, S be two polynomially balanced predicates such that S can be decided in time $O(h(n))$. Suppose that Enum·R and Enum·S are in Delay$(g(n), f(n))$ then, Enum·$(R \cup S)$ is in Delay$(g(n), f(n) + h(n))$.

Proof. Let M_R and M_S be two RAM machines, which solve Enum·R and Enum·S. One builds a machine $M_{(R\cup S)}$ which solves Enum·$(R \cup S)$ by running M_R and M_S in parallel on the instance x. The behavior of $M_{(R\cup S)}$ is described in Algorithm 1.

At each step $M_{(R\cup S)}$ produces a new solution y of $R(x)$ thanks to M_R and it tests if $y \in S(x)$ in time $h(|x|)$, by hypothesis. If $y \notin S(x)$ it outputs it, otherwise it is discarded and the next solution of $S(x)$ given by M_S is computed and outputted[2]. If there is no solution left in $R(x)$ (resp. $S(x)$), it finishes the enumeration thanks to M_R (resp. M_S).

Remark that if $M_{(R\cup S)}$ has enumerated k elements of $S(x)$ thanks to M_S then it has also found and discarded k elements of $R(x) \cap S(x)$ given by M_R. Therefore if $M_{(R\cup S)}$ has outputted all $S(x)$, it has used M_R to produce $|S(x)|$ elements of $R(x) \cap S(x)$, which must then satisfy $S(x) = S(x) \cap R(x)$. Therefore the enumeration of the remaining elements of $R(x)$ does not create any repetition. Moreover all elements of $R(x) \cap S(x)$ are enumerated by M_S only, thus the algorithm makes no repetition.

Since, at each step of the algorithm we simulate M_R and M_S to let them produce at most one solution, the delay of $M_{(R\cup S)}$ is bounded by the sum of the delays of M_R and M_S, that is $2f(|x|)$ plus $h(|x|)$ the time to do one membership test. ◀

▶ **Corollary 9.** *Let $\Phi(\mathbf{y}, T) = \exists \mathbf{x} \varphi(\mathbf{x}, \mathbf{y}, T)$ be a first order formula with $|\mathbf{x}| = k$. Assume that there is an algorithm such that, for all input structures $\mathcal{S}$ of domain D and for all k-tuples $\mathbf{x}^*$ of $\mathcal{S}$, enumerates the elements of $\Phi_{\mathbf{x}^*}(\mathcal{S})$ where $\Phi_{\mathbf{x}^*}(\mathbf{y}, T) = \varphi(\mathbf{x}^*, \mathbf{y}, T)$, with precomputation $g(|D|)$ and delay $f(|D|)$. Then* Enum·Φ *can be computed with a $O(g(|D|)|D|^k)$ precomputation and a delay $O(f(|D|)|D|^k)$.*

Proof. Remark that, for all models $\mathcal{S}$ of domain D, $\Phi(\mathcal{S}) = \cup_{\mathbf{x}^* \in D^k} \Phi_{\mathbf{x}^*}(\mathcal{S})$. We can apply the previous proposition to this union of $|D|^k$ enumerations problems. For each $\mathbf{x}^* \in D^k$, one has to compute $\Phi_{\mathbf{x}^*}(\mathbf{y}, T)$ and do the corresponding precomputation in time $O(g(|D|))$, which accounts for a total precomputation of $O(g(|D|)|D|^k)$. A formula $\Phi_{\mathbf{x}^*}(\mathbf{y}, T)$ is of constant size, therefore checking if $(\mathcal{S}, \mathbf{y}^*, T^*) \models \Phi_{\mathbf{x}^*}(\mathbf{y}, T)$ can be done in constant time. By induction, one can easily generalize Proposition 8 to handle the union of $|D|^k$ predicates. This yields a delay in $O(|D|^k \times f(|D|) + |D|^k) = O(f(|D|)|D|^k)$. ◀

The previous corollary allows to remove the first level of existential quantification of any formula with a polynomial slowdown only. As a consequence, we have a polynomial delay enumeration algorithm for any problem in Enum·$\mathbf{\Sigma}_1$.

▶ **Theorem 10.** Enum·$\mathbf{\Sigma}_1 \subseteq$ DelayP. *More precisely,* Enum·$\mathbf{\Sigma}_1$ *can be computed with precomputation $O(|D|^{h+k})$ and delay $O(|D|^k)$ where h is the number of free first order variables of the formula, k the number of existentially quantified variables and D is the domain of the input structure.*

Proof. Let $\exists \mathbf{x} \varphi(\mathbf{x}, \mathbf{y}, T)$ be a formula of $\mathbf{\Sigma}_1$ and $\mathcal{S}$ be a structure. By Theorem 5, we know that for each k-uple $\mathbf{x}^*$, the solutions of $\varphi(\mathbf{x}^*, \mathbf{y}, T)$ can be enumerated with precomputation $O(|D|^h)$ and a delay $O(1)$. Thus, by Corollary 9, we know that Enum·$\exists \mathbf{x} \varphi(\mathbf{x}, y, T)$ can be computed with precomputation $O(|D|^{h+k})$ and delay $O(|D|^k)$. ◀

[2] note that it can be y itself

Again, for $\mathbf{\Sigma}_1$ queries on structures of bounded degree a better bound can be found at least for the model checking problem. The following holds with a proof similar to (the first steps of) that of Theorem 6.

▶ Proposition 11. Let $d \in \mathbb{N}$. Checking whether $\Phi(\mathcal{S}) = \emptyset$ where $\Phi(\mathbf{z}, T) \in \mathbf{\Sigma}_1$ and $\mathcal{S}$ is a d-degree bounded input structures can be done in time $O(|D|)$ (in data complexity).

It is however open whether the result can be extended in the enumeration setting to prove a linear delay algorithm for this latter kind of query.

3.1 Relation with DNF formulas

In this section, we examine more closely the relationships between the enumeration problem for $\mathbf{\Sigma}_1$-queries and the enumeration of the solutions of restricted DNF formulas. Let ψ be a DNF formula such that each clause is of size at most l. Remark that the number of clauses cannot be larger than n^l where n is the number of variables. We say that such a formula is in DNF(l) and we note Enum·DNF(l) the problem Enum·DNF restricted to DNF(l).

Let now be Φ a formula $\exists \mathbf{x} \varphi(\mathbf{x}, T)$ where T is a second order variable of arity 1 and the tuple $\mathbf{x}$ is such that $|\mathbf{x}| = k$. We also assume that φ is quantifier-free in disjunctive normal form and that each of its clauses contains at most l occurrences of a term involving T. Remark that $l \leq k$, because each occurrence of T in a clause must be applied to a different variable. On the other hand, one can rename the l variables used in one clause without changing the satisfying assignments of $\exists \mathbf{x} \varphi(\mathbf{x}, T)$. Therefore, we can use the same l variables in each clause and delete the others to obtain an equivalent formula with l variables. Thus, the parameters k and l are essentially the same. We denote by $\Sigma_1(l)$ the set of such formulas (with $k = l$).

▶ Remark. Here we do not allow free first-order variables. It is always possible to take care of them with a polynomial slowdown in the precomputation only.

Moreover, the restriction on the arity of the second order variable could be lifted and we would obtain essentially the same results. We choose this restriction, because in this setting, we have the parameters k and l equal which makes the next propositions easier to state and to understand.

▶ Proposition 12. If Enum·DNF(l) can be solved with precomputation $g(n)$ and delay $f(n)$, where n is the number of variables of the formula, then for all formulas $\Phi \in \Sigma_1(l)$, Enum·Φ can be solved with precomputation $g(n)$ and delay $f(n)$, where n is the size of the domain.

Proof. To prove that, fix a formula $\Phi \equiv \exists \mathbf{x} \varphi(\mathbf{x}, T) \in \Sigma_1(l)$. Let $\mathcal{S}$ be the input structure and D its domain. Let $\tilde{\Phi}$ be the propositional formula associated to Φ as before. It is the disjunction of the n^l formulas $\Phi(\mathbf{x}^*, T)$. Each of the formula $\Phi(\mathbf{x}^*, T)$ is a DNF formula with clauses of size at most l. Therefore the formula $\tilde{\Phi}$ is in DNF(l), has $|D|$ variables and its solutions are in bijection with the solutions of Φ. ◀

▶ Proposition 13. There is a formula Φ in $\Sigma_1(l, l)$ such that the following holds. If Enum·Φ can be solved with precomputation $g(n)$ and delay $f(n)$, where n is the size of the domain, then Enum·DNF(l) can be solved with precomputation $g(n)$ and delay $f(n)$, where n is the number of variables of the formula.

Proof. Let σ be the language $\{P_{i,j}\}_{i+j \leq l}$, where $P_{i,j}$ is an l-ary predicate. A predicate $P_{i,j}$ represents, in the reduction, a clause whose first i variables appear positively and the next j appear negatively. The second order variable T represents the set of variables set to true. Let

$$\theta_{i,j}(T, x_1, \ldots, x_l) \equiv \bigwedge_{s \leq i} x_s \in T \wedge \bigwedge_{i < s \leq l} x_s \notin T$$

and let

$$\Phi \equiv \exists x_1, \ldots, x_l \bigvee_{i+j \leq l} (P_{i,j}(x_1, \ldots, x_l) \wedge \theta_{i,j}(T, x_1, \ldots, x_l)).$$

Let now ψ be a DNF formula over the variables $V = \{v_1, \ldots, v_n\}$. We reduce the enumeration of the solutions of ψ to the enumeration of $\Phi(\mathscr{S})$. The domain of $\mathscr{S}$ is the set V and $P_{i,j}(x_1, \ldots, x_l)$ holds if and only if there is a clause in ψ whose variables appearing positively are $x_1, \ldots, x_i$ and those appearing negatively are $x_{i+1}, \ldots, x_{i+j}$. Remark now that $T^* \in \Phi(\mathcal{S})$ if and only if T^* represents an assignment of the variables in V which satisfies ψ. Thus the solutions of ψ are in bijection with $\Phi(\mathcal{S})$ which achieves the proof. ◀

The above propositions justify in some sense why the enumeration complexity of $\mathbf{\Sigma}_1$ queries and Enum·DNF are intimately related. Hence, to improve our results on Enum·$\mathbf{\Sigma}_1$, one has to study the problem Enum·DNF(l). The following question seems quite challenging:

Open Question: prove (or disprove) that there exists an enumeration algorithm for Enum·DNF(l) whose delay does not depend on l or at least is better than $O(n^l)$.

4 Enumeration for $\mathbf{\Pi}_1$ formulas and beyond

In [12], it is shown that the propositional satisfiability problem for a 3-CNF formula can be expressed as a query problem for a $\mathbf{\Pi}_1$ formula. The following result then holds.

▶ Proposition 14. Unless P = NP, there is no polynomial delay algorithm for Enum·$\mathbf{\Pi}_1$. The results still hold even for structure of bounded degree as input.

Proof. See [12]. For the case of bounded degree structures, remark that it is well-known that the satisfiability problem is hard even for 3-CNF formulas such that each variable appears (positively or negatively) in at most 3 clauses. For such propositional formulas, the structures obtained after reduction in [12] is of bounded degree. ◀

As it is shown below, it is even possible to define the satisfiability problem by a quite restricted $\mathbf{\Pi}_1$ formula. A 3-CNF formula φ can be encoded by a structure $\mathcal{S}_\varphi$ of signature $\{C, a_1, a_2, a_3, a_4\}$ where C is a 4-ary predicate and a_1, a_2, a_3, a_4 are constants. The domain of $\mathcal{S}_\varphi$ contains as many elements as variables in φ. Let x, y and z be elements of the domain, $C(a_i, x, y, z,)$ is true if the clause $\neg_{i,1}x \vee \neg_{i,2}y \vee \neg_{i,3}z$ appears in φ where $\neg_{i,j} = \neg$ if $i \leq j$ (and $\neg_{i,j} = \epsilon$ if not). In other words, i encodes the number of variables that appear negatively in the clause. Let $\Psi(T, T_1, T_2, T_3)$ be the following $\mathbf{\Pi}_1$ formula:

$$\begin{array}{l} \forall x_1 \forall x_2 \forall x_3 \forall a \\ (C(a, x_1, x_2, x_3) \rightarrow T_1(a, x_1) \vee T_2(a, x_2) \vee T_3(a, x_3)) \wedge \\ \bigwedge_{i=1}^{4} \bigwedge_{j=1}^{3} T_j(a_i, x_1) \leftrightarrow \neg_{i,j} T(x_1) \end{array} \quad (2)$$

Clearly, there is a bijective correspondence between the satisfying assignments of φ and the set $\Psi(\mathcal{S}_\varphi)$. Remark now that the quantifier free part of Ψ is in CNF and is such that all its clauses except one have at most two occurences of a second-order free variable.

In [12], a first-order formula Φ defines the problem of computing the cardinal of $\Phi(\mathcal{S})$. Theorem 2 of [12] describes the strict inclusions of the classes of counting functions defined

by the number of quantifier alternations. It can be easily transposed into the following theorem on enumeration problems, if we assume the set of all models to have a total order.

▶ **Theorem 15.** *On linearly ordered structures, the following inclusions hold:*

$$\textsc{Enum}\cdot\mathbf{\Sigma}_0 \subsetneq \textsc{Enum}\cdot\mathbf{\Sigma}_1 \subsetneq \textsc{Enum}\cdot\mathbf{\Pi}_1 \subsetneq \textsc{Enum}\cdot\mathbf{\Sigma}_2 \subsetneq \textsc{Enum}\cdot\mathbf{\Pi}_2.$$

Moreover, there is a problem in $\textsc{Enum}\cdot\mathbf{\Pi}_2$ which is complete up to parsimonious reduction for all problems definable by a polynomially balanced predicate (a polynomial time reduction f between two decision problems A and B is *parsimonious* if, for each valid instance x, it establishes a bijective correspondence between the solutions sets $A(x)$ and $B(f(x))$. See, for example, [12] for a precise definition). Therefore the hierarchy collapses at $\textsc{Enum}\cdot\mathbf{\Pi}_2$.

4.1 Feasible classes beyond $\mathbf{\Sigma}_1$

We now consider fragments of $\mathbf{\Pi}_2$ and $\mathbf{\Sigma}_2$ with a good expressive power and whose associated enumeration problems remain tractable. Let $\mathcal{C}$ be a subclass of propositional formulas.

$\textsc{Enum}\cdot\textsc{SAT}(\mathcal{C})$
Input: A propositional CNF formula φ in $\mathcal{C}$
Output: an enumeration of the satisfying assignments of φ.

A CNF formula is Horn (resp. anti-Horn) if it is equivalent to a formula whose clauses have at most one positive (resp. negative) literal. It is bijunctive if it is equivalent to a CNF formula with clauses of length at most two. Finally, it is affine if it is equivalent to a system of linear equations over the two-element field. We first investigate the immediate consequence of the following result and the fact that to solve $\textsc{Enum}\cdot\Phi$, one only has to enumerate the solutions obtained from those of $\tilde{\Phi}$ as in Theorem 5.

▶ Proposition 16 ([5]). The problem $\textsc{Enum}\cdot\textsc{SAT}(\mathcal{C})$ is in DelayP when $\mathcal{C}$ is one of the following classes: Horn formulas, anti-Horn formulas, affine formulas, bijunctive (2-CNF) formulas

▶ **Corollary 17.** *Let $\Phi(\mathbf{z}, T)$ be a formula, such that, for all σ structures, all propositional formulas $\tilde{\Phi}_i$ are either Horn, anti-Horn, affine or bijunctive. Then* $\textsc{Enum}\cdot\Phi \in \textsc{DelayP}$.

Proof. Let $\Phi(\mathbf{z}, T)$ be a formula, with $|\mathbf{z}| = k$ and $\mathcal{S}$ a structure of domain D. Let $\mathbf{z}_i$ be an enumeration of the k-tuples of D. Recall that the set of solutions of $\Phi(\mathbf{z}_i, T)$ is equal to:

$$\bigcup_{I \in \mathcal{I}(\Phi_i)} \bigcup_{T^* \in [T(\mathbf{z}_i, I)]} (\mathbf{z}_i, T^*).$$

Furthermore, we know that $\tilde{\Phi}_i$ is either Horn, anti-Horn, affine or bijunctive and that it is a $\mathbf{\Pi}_1$ formula. By construction, it is of size polynomial in $|D|$, hence its models can be enumerated in polynomial delay by Proposition 16. The enumeration of the solutions of $\textsc{Enum}\cdot\Phi$ on the model $\mathcal{S}$ is done in polynomial delay as follows:

- for each $\mathbf{z}_i$ compute the formula $\tilde{\Phi}_i$
- enumerate the models of each $\tilde{\Phi}_i$ in polynomial delay
- for each model I of $\tilde{\Phi}_i$, build in polynomial time the solution $(\mathbf{z}_i, T(\mathbf{z}_i, I))$
- for each solution $(\mathbf{z}_i, T(\mathbf{z}_i, I))$, generate by Gray code enumeration the solutions $(\mathbf{z}_i, T^*)$ with $T^* \in [T(\mathbf{z}_i, I)]$

Remark that for two different $\mathbf{z}_i$ the enumerated solutions are disjoint.

Moreover, for $I \neq J$, $T(\mathbf{z}_i, I)$ and $T(\mathbf{z}_i, J)$ differs on at least one value. Hence, $T(\mathbf{z}_i, I) \cap T(\mathbf{z}_i, J) = \emptyset$. Therefore, there are no repetition in the previously described algorithm. ◀

The condition in Corollary 17 is semantic: it applies to $\tilde{\Phi}$ and not to Φ, which makes it not obvious to characterize. The following result holds in contrast with the case of Formula (2) which shows that $\mathbf{\Pi}_1$ queries in conjunctive normal form that have one clause with three occurrences of a second order variable are hard to enumerate.

▶ **Corollary 18.** *Let $\Phi(\mathbf{z}, T) \equiv \exists \mathbf{y} \forall \mathbf{x} \Psi(\mathbf{x}, \mathbf{y}, \mathbf{z}, T)$ where Ψ is in conjunctive normal form and all its clauses contain at most 2 occurrences of a free predicate then* Enum·$\Phi \subseteq$ DelayP.

Proof. Let $\mathcal{S}$ be a finite structure and $\Phi(\mathbf{z}, T)$ as above. For such a $\Phi(\mathbf{z}, T)$, the formula $\tilde{\Phi}_i$ is of the form (set $|y| = p$)

$$\bigvee_{j=1}^{n^p-1} \Psi_i(\mathbf{z}_i, \mathbf{y}_j)$$

where $\Psi_i(\mathbf{z}_i, \mathbf{y}_j)$ is a 2-CNF formula of size polynomial in $|S|$. From Proposition 16, models of such formulas can be enumerated with polynomial delay. The union of models of the polynomially many formulas $\Psi_i(\mathbf{z}_i, \mathbf{y}_j)$ can be enumerated following the method of Proposition 8. The delay is then polynomial. ◀

The above corollary applies to $R\Sigma_2$ formulas defined in [12]. It has been shown there that counting the models of such formulas can be done by a fully polynomial randomized approximation scheme.

▶ **Example 19.** The formula $IS(T) \equiv \forall x \forall y T(x) \wedge T(y) \Rightarrow \neg E(x, y)$ satisfies the condition of the previous corollary therefore Enum·IS $\in$ DelayP. Some other interesting objects such as vertex covers can be defined by a formula of this form.

The next result is similar to Corollary 18, but in its proof it uses a Horn or an anti-Horn formula instead of a 2 CNF formula.

▶ **Corollary 20.** *Let $\Phi(z, T) \equiv \forall x \exists y \Psi_1(x, y, z, T)$ where Ψ is in disjunctive normal form such that each of its clauses contain only one occurrence of a free second order variable and all these occurrences are of the same polarity. Then* Enum·$\Phi \subseteq$ DelayP.

▶ **Example 21.** The formula $DS(T) \equiv \forall x \exists y T(t) \wedge E(x, y)$ holds if and only if T is a dominating set. Since $DS(T)$ satisfies the hypothesis of Corollary 20, Enum·DS $\in$ DelayP.

▶ **Example 22.** Recall that $HS(T) \equiv \forall x (T(x) \Rightarrow V(x)) \wedge \forall y \exists x E(y) \Rightarrow (T(x) \wedge R(x, y))$ characterizes the hitting sets of an hypergraph. It does not exactly satisfy the hypothesis of the previous corollary because T appears with a different polarity in $\forall x (T(x) \Rightarrow V(x))$ and in $\forall y \exists x E(y) \Rightarrow (T(x) \wedge R(x, y))$. However, if we consider the formula $\tilde{HS}(T)_i$, we see that it is a Horn formula which enables us to conclude by Corollary 17 that Enum·HS $\in$ DelayP.

5 Concluding remarks

The results of this paper try to give a first overview of the complexity of first order query problems with possibly free second order variables. Not surprisingly, the complexity increases rapidly with alternation of quantifiers: if the first levels $\mathbf{\Sigma}_0$ and $\mathbf{\Sigma}_1$ admit efficient enumeration algorithm (with constant or polynomial delay), the $\mathbf{\Pi}_1$ is already able to express hard problems. However, some interesting subcases beyond $\mathbf{\Sigma}_1$ are exhibited which admit rather efficient enumeration algorithms.

An interesting question is whether one can extend our result for first order logic with additional operators (such as fixpoint or maximization/minimization operators). Among them, let Enum·$MaxT\Phi(T)$ be the problem of enumerating all maximal models of φ.

It is easy to see that if Φ satisfies the hypotheses of Corollary 18, then Enum·$MaxT\varphi(T)$ is in DelayP by a result of [9] (this case captures among other things the problem of enumerating the maximal (for inclusion) independent sets of a graph). On the other hand, since enumerating the maximal models of a Horn formula is hard (see [9] also), obtaining such a result when hypotheses of Corollary 20 are satisfied seems very unlikely.

References

1 G. Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL*, volume 4207, pages 167–181, 2006.

2 G. Bagan. *Algorithmes et Complexité des Problèmes d'Énumération pour l'Évaluation de Requêtes Logiques*. PhD thesis, Université de Caen, 2009.

3 G. Bagan, A. Durand, and E. Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2007.

4 B. Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics*, 157(12):2675–2700, 2009.

5 N. Creignou and J.J. Hébrard. On generating all solutions of generalized satisfiability problems. *RAIRO Theoretical Informatics and Applications*, 31(6), 1997.

6 A. Durand and E. Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Transactions on Computational Logic (TOCL)*, 8(4), 2007.

7 R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *American Mathematical Society*, pages 43–74, 1974.

8 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

9 D.J. Kavvadias, M. Sideri, and E.C. Stavropoulos. Generating all maximal models of a Boolean expression. *Information Processing Letters*, 74(3-4):157–162, 2000.

10 L. Libkin. *Elements of finite model theory*. EATCS Series. Springer, 2004.

11 S. Lindell. A normal form for first-order logic over doubly-linked data structures. *International Journal of Foundations of Computer Science*, 19(1):205–217, 2008.

12 S. Saluja, K.V. Subrahmanyam, and M.N. Thakur. Descriptive complexity of# P functions. *Journal of Computer and System Sciences*, 50(3):493–505, 1995.

13 Y. Strozecki. *Enumeration complexity and matroid decomposition*. PhD thesis, Université Paris Diderot - Paris 7, 2010.

On Constraint Satisfaction Problems below P*

László Egri

School of Computer Science, McGill University
Montreal, Canada
laszlo.egri@mail.mcgill.ca

Abstract

Symmetric Datalog, a fragment of the logic programming language *Datalog*, is conjectured to capture all constraint satisfaction problems (CSP) in logarithmic space [10]. Therefore developing tools that help us understand whether or not a CSP can be defined in symmetric Datalog is an important task. A simple, well-known fact is that for any CSP, a fixed set of structures $\mathcal{O}$ (an *obstruction set*) can be defined such that a CSP instance I is a yes-instance iff no structure in $\mathcal{O}$ *maps homomorphically* to I. A CSP having X*-duality* means that the set $\mathcal{O}$ can be chosen to have property X. It is widely known that a CSP is definable in Datalog and *linear* Datalog iff that CSP has *bounded treewidth* [12] and *bounded pathwidth* duality [6], respectively. In the case of symmetric Datalog, Bulatov, Krokhin and Larose ask for such a duality in [4]. We provide two such dualities, and we give applications. In particular, we give a short and simple new proof of the main result of [8] that "Maltsev + Datalog ⇒ symmetric Datalog".

In the second part of the paper, we provide some evidence for the conjecture that every CSP in nondeterministic logarithmic space (NL) is definable in the Datalog fragment linear Datalog [6]. We recall that every problem in NL can be defined by a linear Datalog program with negation and access to an order over the domain of its input (linDat($\mathfrak{suc}$,¬)) [6, 13, 15], or by a poly-size family of *nondeterministic branching programs* [20]. We consider the following restrictions of the previous models: *read-once* linDat($\mathfrak{suc}$) (1-linDat($\mathfrak{suc}$)), and *monotone read-once* nondeterministic branching programs (mnBP1). Although restricted, these models can still define NL-complete problems such as directed *st*-Connectivity, *and also nontrivial problems in* NL *which are not definable in linear Datalog.* We show that any CSP definable by a 1-linDat($\mathfrak{suc}$) program or by a poly-size family of mnBP1s can also be defined by a linear Datalog program. It also follows that a wide class of CSPs–CSPs which do not have bounded pathwidth duality (e.g. the P-complete Horn-3Sat problem)–cannot be defined by any 1-linDat($\mathfrak{suc}$) program or by any poly-size family of mnBP1s.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Constraint satisfaction problems, complexity classes, Datalog fragments

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.203

1 Introduction

Constraint satisfaction problems (CSP) constitute a unifying framework to study various computational problems arising naturally in various branches of computer science, including artificial intelligence, graph homomorphisms, and database theory. Loosely speaking, an instance of a CSP consists of a list of variables and a set of constraints, each specified by an ordered tuple of variables and a constraint relation over some specified domain. The goal is

* Research supported by the Natural Sciences and Engineering Research Council of Canada (NSERC). We thank Benoit Larose and Pascal Tesson for useful discussions and comments. We also thank the anonymous referees for their in-depth reviews.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 203–217

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

then to determine whether variables can be assigned domain values such that all constraints are simultaneously satisfied.

Recent efforts have been directed at classifying the complexity of the so-called *nonuniform* CSP. For a fixed finite set of finite relations Γ, CSP(Γ) denotes the nonuniform CSP corresponding to Γ. The difference between an instance of CSP(Γ) and an instance of the general CSP is that constraints in an instance of CSP(Γ) take the form $(x_{i_1}, \ldots, x_{i_k}) \in R$ for some $R \in \Gamma$. Examples of nonuniform CSPs include k-Sat, Horn-3Sat, Graph H-Coloring, and many others.

For a relational structure $\mathbf{B}$, the homomorphism problem HOM($\mathbf{B}$) takes a structure $\mathbf{A}$ as input, and the task is to determine if there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$. For instance, consider structures that contain a single symmetric binary relation, i.e. graphs. A homomorphism from a graph $\mathbf{G}$ to a graph $\mathbf{H}$ is a mapping from $V_{\mathbf{G}}$ to $V_{\mathbf{H}}$ such that any edge of $\mathbf{G}$ is mapped to an edge of $\mathbf{H}$. If $\mathbf{H}$ is a graph with a single edge then HOM($\mathbf{H}$) is the set of graphs which are two-colorable. There is a well-known and straightforward correspondence between the CSP and the homomorphism problem. For this reason, from now on we work only with the homomorphism problem instead of the CSP. Nevertheless, we call HOM($\mathbf{B}$) a CSP and we also write CSP($\mathbf{B}$) instead of HOM($\mathbf{B}$), as it is often done in the literature.

The CSP is of course NP-complete, and therefore research has focused on identifying "islands" of tractable CSPs. The well-known CSP dichotomy conjecture of Feder and Vardi [12] states that every CSP is either tractable or NP-complete, and progress towards this conjecture has been steady during the last fifteen years. From a complexity-theoretic perspective, the classification of CSP($\mathbf{B}$) as in P or being NP-complete is rather coarse and therefore somewhat dissatisfactory. Consequently, understanding the fine-grained complexity of CSPs gained considerable attention during the last few years. Ultimately, one would like to know the precise complexity of a CSP lying in P, i.e. to identify a "standard" complexity class for which a given CSP is complete. Towards this, it was established that Schaefer's P – NP dichotomy for Boolean CSPs [19] can indeed be refined: each CSP over the Boolean domain is either definable in first order logic, or complete for one of the classes L, NL, $\oplus$L, P or NP under AC^0-*reductions* [2]. The question whether some form of this fine-grained classification extends to non-Boolean domains is rather natural. The two most important tools to study CSPs whose complexity is below P are *symmetric Datalog* and *linear Datalog*, syntactic restrictions of the database-inspired logic programming language *Datalog*. We say that co-CSP($\mathbf{B}$)–the complement of CSP($\mathbf{B}$)–is definable in (linear, symmetric) Datalog if the set of structures that do not homomorphically map to $\mathbf{B}$ is accepted by a (linear, symmetric) Datalog program.[1]

Symmetric Datalog programs can be evaluated in logarithmic space (L), and in fact, it is conjectured that if co-CSP($\mathbf{B}$) is in L then it can also be defined in symmetric Datalog [10]. There is a considerable amount of evidence supporting this conjecture (see, for example, [10, 9, 8, 16, 5]), and therefore providing tools to show whether co-CSP($\mathbf{B}$) can be defined in symmetric Datalog is an important task. It is well known and easy to see that for any structure $\mathbf{B}$, there is a set of structures $\mathcal{O}$, called an *obstruction set*, such that a structure $\mathbf{A}$ *homomorphically maps to* $\mathbf{B}$ iff there is no structure in $\mathcal{O}$ that homomorphically maps to $\mathbf{A}$. In fact, there are many possible obstruction sets for any structure $\mathbf{B}$. We say that $\mathbf{B}$ has duality X, if $\mathbf{B}$ has an obstruction set which has the special property X. The following two well-known theorems relate definability of co-CSP($\mathbf{B}$) in Datalog and linear Datalog to $\mathbf{B}$

[1] The reason we define co-CSP($\mathbf{B}$) instead of CSP($\mathbf{B}$) in (linear, symmetric) Datalog is a technicality explained in Section 2.5.

having *bounded treewidth* duality and *bounded pathwidth duality*, respectively:

1. co-CSP($\mathbf{B}$) is definable in Datalog iff $\mathbf{B}$ has bounded treewidth duality [12];
2. co-CSP($\mathbf{B}$) is definable in linear Datalog iff $\mathbf{B}$ has bounded pathwidth duality [6].

It was stated as an open problem in [4] to find a duality for symmetric Datalog in the spirit of the previous two theorems. We provide two such dualities: *symmetric bounded pathwidth duality* (SBPD) and *piecewise symmetric bounded pathwidth duality* (PSBPD). We note that SBPD is a special case of PSBPD. For both bounded treewidth and bounded pathwidth duality, the structures in the obstruction sets are restricted to have some special form. For SBPD and PSBPD the situation is a bit more subtle. In addition that we require the obstruction sets to contain structures only of a special form (they must have bounded pathwidth), the obstruction sets must also possess a certain "symmetric closure" property. To the best of our knowledge, this is the first instance of a duality where in addition to the local requirement that each structure must be of a certain form, the set must also satisfy an interesting global requirement.

Using SBPD, we give a short and simple new proof of the main result of [8] that "Maltsev + Datalog $\Rightarrow$ symmetric Datalog". Considering the simplicity of this proof, we suspect that SBPD (or PSBPD) could be a useful tool in an attempt to prove the *symmetric Datalog conjecture* [16], a conjecture that proposes an algebraic characterization of all CSPs lying in L. An equivalent form of this conjecture is that "Datalog + n-permutability $\Rightarrow$ symmetric Datalog" (by combining results from [14],[3] and [17]), where n-permutability is a generalization of Maltsev.

One way to gain more insight into the dividing line between CSPs in L and NL is through studying the complexity of CSPs corresponding to oriented paths. The only known thing regarding the complexity of these CSPs is that they are all in NL (by combining results from [11, 7, 6]). To make progress in this direction, it is natural to ask whether there are oriented paths for which the CSP is NL-complete and L-complete. We provide two classes of oriented paths, $\mathcal{C}_1$ and $\mathcal{C}_2$, such that for any $\mathbf{B}_1 \in \mathcal{C}_1$, the corresponding CSP is NL-complete, and for any $\mathbf{B}_2 \in \mathcal{C}_2$, the corresponding CSP is L. In fact, it can be seen with the help of [16] that for most $\mathbf{B}_2 \in \mathcal{C}_2$, CSP($\mathbf{B}_2$) is L-complete. To prove the membership of CSP($\mathbf{B}_2$) in L (for $\mathbf{B}_2 \in \mathcal{C}_2$), we use PSBPD in an essential way. One can hope to build on this work to achieve an L-NL dichotomy for oriented paths.

In the second part of the paper, we investigate CSPs in NL. Based on the observation that any CSP known to be in NL is also known to be definable by a linear Datalog program, Dalmau conjectured that every CSP in NL can be defined by a linear Datalog program [6]. Linear Datalog($\mathfrak{suc}$,$\neg$) (linDat($\mathfrak{suc}$,$\neg$)) denotes the extension of linear Datalog in which we allow negation and access to an order over the domain of the input. It is known that any problem in NL can be defined by a linDat($\mathfrak{suc}$,$\neg$) program [6, 13, 15], and therefore one way to prove the above conjecture would be to show that any CSP that can be defined by a linDat($\mathfrak{suc}$,$\neg$) program can also be defined by a linear Datalog program. We consider a restriction of the conjecture because proving it in its full generality would separate NL from P (using [1]).

Read-once linear Datalog($\mathfrak{suc}$) (1-linDat($\mathfrak{suc}$)) is a subclass of linDat($\mathfrak{suc}$,$\neg$), but a subclass that has interesting computational abilities, and for which we are able to find the chink in the armor. We can easily define some NL-complete problems in 1-linDat($\mathfrak{suc}$), such as the CSP directed *st*-connectivity (*st*-Conn), and also problems that are not *homomorphism-closed*, such as determining if the input graph is a clique on 2^n vertices, $n \geq 1$. Because any problem that can be defined with a linear Datalog program must be homomorphism closed, it follows that 1-linDat($\mathfrak{suc}$) can define nontrivial problems which are in NL but which are not definable

by any linear Datalog program. However, our main result shows that if co-CSP($\mathbf{B}$) can be defined by a 1-linDat($\mathfrak{suc}$) program, then co-CSP($\mathbf{B}$) can also be defined by a linear Datalog program. The crux of our argument applies the general case of the Erdős-Ko-Rado theorem to show that a 1-linDat($\mathfrak{suc}$) program does not have enough "memory" to handle structures of unbounded pathwidth.

Our proof establishing the above result for 1-linDat($\mathfrak{suc}$) programs can be adapted to show a parallel result for a subclass of *nondeterministic branching programs*, which constitute an important and well-studied class of computational models (see the book [20]). More precisely, we show that if co-CSP($\mathbf{B}$) can be defined by a *poly-size family of read-once*[2] *monotone nondeterministic* branching programs (mnBP1(poly)) then co-CSP($\mathbf{B}$) can also be defined by a linear Datalog program.[3]

Finally, our results can be interpreted as lower-bounds on a wide class of CSPs: if $\mathbf{B}$ does not have bounded pathwidth duality, then co-CSP($\mathbf{B}$) cannot be defined with any 1-linDat($\mathfrak{suc}$) program or with any mnBP1(poly). A specific example of such a CSP would be the P-complete Horn-3Sat problem, and more generally, Larose and Tesson showed that any CSP whose associated *variety admits the unary, affine or semilattice types* does not have bounded pathwidth duality (see [16] for details).

2 Preliminaries

2.1 Algebra

A *vocabulary* (or *signature*) is a finite set of relation symbols with associated arities. The arity function is denoted with $\mathrm{ar}(\cdot)$. If $\mathbf{A}$ is a relational structure over a vocabulary τ, then $R^{\mathbf{A}}$ denotes the relation of $\mathbf{A}$ associated with the symbol $R \in \tau$. The lightface equivalent of the name of the structure denotes the universe of the structure, e.g. the universe of $\mathbf{A}$ is A.

A *tuple structure* $\tilde{\mathbf{A}}$ over a vocabulary τ is a set of pairs $(R, \mathbf{t})$ where $R \in \tau$ and $\mathbf{t}$ is an $\mathrm{ar}(R)$-tuple. We associate a domain $\tilde{A}$ with a tuple structure: $\tilde{A}$ contains every element that appears in some tuple in $\mathbf{A}$, and possibly some other elements. Clearly, tuple structures are equivalent to relational structures. If $\mathbf{A}$ is a relational structure, we denote the equivalent tuple structure with $\tilde{\mathbf{A}}$, and vice versa. For convenience, we use the two notations interchangeably. We note that *all* structures in this paper are finite.

Let $\mathbf{B}$ be a structure of the same signature as $\mathbf{A}$. A *homomorphism* from $\mathbf{A}$ to $\mathbf{B}$ is a map f from A to B such that $f(R^{\mathbf{A}}) \subseteq R^{\mathbf{B}}$ for each $R \in \tau$. A structure is called a *core* if it has no homomorphism to any of its proper substructures. If there exists a homomorphism from $\mathbf{A}$ to $\mathbf{B}$, we often denote it with $\mathbf{A} \to \mathbf{B}$. If that homomorphism is f, we write $\mathbf{A} \xrightarrow{f} \mathbf{B}$. We denote by CSP($\mathbf{B}$) the class of all τ-structures $\mathbf{A}$ such that $\mathbf{A} \to \mathbf{B}$, and by co-CSP($\mathbf{B}$) the complement of CSP($\mathbf{B}$). If we are given a class of τ-structures $\mathcal{C}$ such that for any $\mathbf{A} \in \mathcal{C}$, and any $\mathbf{B}$ such that $\mathbf{A} \to \mathbf{B}$ it holds that $\mathbf{B} \in \mathcal{C}$, then we say that $\mathcal{C}$ is *homomorphism-closed*. *Isomorphism closure* is defined in a similar way.

An n-ary operation on a set A is a map $f : A^n \to A$. Given an h-ary relation R and an n-ary operation f on the same set A, we say that f *preserves* R or that R is *invariant* under f if the following holds: given any matrix M of size $h \times n$ whose columns are in R,

[2] Our read-once restriction for nondeterministic branching programs is less stringent than the usual definition because we require the programs to be read-once only on certain inputs.

[3] A 1-linDat($\mathfrak{suc}$) can be converted into an mnBP1(poly), so another way to present our results would be to do the proofs in the context of mnBP1s, and then to conclude the parallel result for 1-linDat($\mathfrak{suc}$).

applying f to the rows of M produces an h-tuple in R. A *polymorphism* of a structure $\mathbf{B}$ is an operation f that preserves each relation in $\mathbf{B}$.

▶ **Definition 1** (Maltsev Operation)**.** A ternary operation $f : A^3 \to A$ on a finite set A is called Maltsev if it satisfies the following identities: $f(x, y, y) = f(y, y, x) = x, \forall x, y \in A$.

2.2 Datalog

We provide only an informal introduction to Datalog and its fragments, and the reader can find more details, for example, in [18, 6, 10]. Datalog is a database-inspired query language whose connection with CSP-complexity is now relatively well understood (see e.g. [3]). Let τ be some finite vocabulary. A Datalog program over τ is specified by a finite set of rules of the form $h \leftarrow b_1 \wedge \cdots \wedge b_t$, where h and the b_i are atomic formulas $R(x_1, \ldots, x_k)$. When we specify the variables of an atomic formula, we always list the variables from left to right, or we simply provide a tuple $\mathbf{x}$ of variables whose i-th variable is $\mathbf{x}[i]$. We distinguish two types of relational predicates occurring in a Datalog program: predicates I that occur at least once in the head of a rule (i.e., its left-hand side) are called *intensional database predicates* (*IDBs*) and are not in τ. The predicates which occur only in the body of a rule (its right-hand side) are called *extensional database predicates* (*EDBs*) and must all lie in τ. A rule that contains no IDB in the body is called a *nonrecursive rule*, and a rule that contains at least one IDB in the body is called a *recursive rule*. A Datalog program contains a distinguished IDB of arity 0 which is called the *goal predicate*; a rule whose head IDB is a goal IDB is called a *goal rule*.

Linear Datalog is a syntactic restriction of Datalog in which there is at most one IDB in the body of each rule. The class of linear Datalog programs that contains only rules with at most k variables and IDBs with at most $j \leq k$ variables is denoted with *linear* (j, k)*-Datalog*. We say that the *width* of such a linear Datalog program is (j, k).

Symmetric Datalog is a syntactic restriction of linear Datalog. A linear Datalog program $\mathcal{P}$ is symmetric if for any recursive rule $I(\mathbf{x}) \leftarrow J(\mathbf{y}) \wedge \bar{E}(\mathbf{z})$ of $\mathcal{P}$ (except for goal rules), where $\bar{E}(\mathbf{z})$ is a shorthand for the conjunction of the EDBs of the rule over variables in $\mathbf{z}$, the symmetric pair $J(\mathbf{y}) \leftarrow I(\mathbf{x}) \wedge \bar{E}(\mathbf{z})$ of that rule is also in $\mathcal{P}$. The *width* of a symmetric Datalog program is defined similarly to the width of a linear Datalog program.

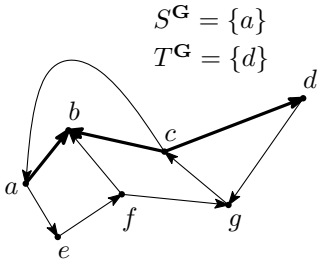
We explain the semantics of linear (symmetric) Datalog using derivations (it could also be explained with *fixed point operators*, but that would be inconvenient for the proofs). Let $\mathcal{P}$ be a linear Datalog program with vocabulary τ. A *$\mathcal{P}$-derivation with codomain D* is a sequence of pairs $\mathscr{D} = (\rho_1, \lambda_1), \ldots, (\rho_q, \lambda_q)$, where ρ_ℓ is a rule of $\mathcal{P}$, and λ_ℓ is a function from the variables V_ℓ of ρ_ℓ to D, $\forall \ell \in [q]$. The sequence $\mathscr{D}$ must satisfy the following properties. Rule ρ_1 is nonrecursive, and ρ_q is a goal rule. For all $\ell \in [q-1]$, the head IDB I of ρ_ℓ is the IDB in the body of $\rho_{\ell+1}$, and if the variables of I in the head of ρ_ℓ and the body of $\rho_{\ell+1}$ are $\mathbf{x}$ and $\mathbf{y}$, respectively, then $\lambda_\ell(\mathbf{x}[i]) = \lambda_{\ell+1}(\mathbf{y}[i])$, $\forall i \in [\mathrm{ar}(I)]$.

Let $R(\mathbf{z})$ be an EDB with variables in some rule ρ_ℓ of a derivation $\mathscr{D}$. Then we write $R(\mathbf{t})$ to denote that $\lambda_\ell(\mathbf{z}) = \mathbf{t}$, i.e. that λ_ℓ *instantiates* the variables of $R(\mathbf{z})$ to $\mathbf{t}$, and we say that $R(\mathbf{t})$ *appears* in ρ_ℓ, or less specifically, that $R(\mathbf{t})$ *appears* in $\mathscr{D}$. Given a structure $\mathbf{A}$ and a derivation $\mathscr{D}$ with codomain A for a program $\mathcal{P}$, we say that $\mathscr{D}$ *is a derivation for* $\mathbf{A}$ if for every $R(\mathbf{t})$ that appears in a rule of $\mathscr{D}$, $(R, \mathbf{t}) \in \tilde{\mathbf{A}}$. We denote a $\mathcal{P}$-derivation for a structure $\mathbf{A}$ with $\mathscr{D}_{\mathcal{P}}(\mathbf{A})$. A linear (symmetric) Datalog program $\mathcal{P}$ *accepts* an input structure $\mathbf{A}$ if there exists a $\mathcal{P}$-derivation for $\mathbf{A}$.

▶ **Definition 2** (Read-Once Derivation)**.** We say that a derivation $\mathscr{D}$ is *read-once* if every $R(\mathbf{t})$ that appears in $\mathscr{D}$ appears exactly once in $\mathscr{D}$, except when R is the special EDB $\mathfrak{suc}$, $\mathfrak{first}$, or $\mathfrak{last}$, defined in Section 4.

An example is given in Fig. 1. The vocabulary is $\tau = \{E^2, S^1, T^1\}$, where the superscripts denote the arity of the symbols. Notice that in the symmetric Datalog program $\mathcal{P}$, rules of types 2 and 3 form a symmetric pair. It is not difficult to see that $\mathcal{P}$ accepts a τ-structure $\mathbf{A}$ iff there is an oriented path (see Section 3.1) in $E^{\mathbf{A}}$ from an element in $S^{\mathbf{A}}$ to an element in $T^{\mathbf{A}}$.

$$
\begin{aligned}
I(x) &\leftarrow S(x) && (1)\\
I(y) &\leftarrow I(x) \wedge E(x,y) && (2)\\
I(x) &\leftarrow I(y) \wedge E(x,y) && (3)\\
G &\leftarrow I(x) \wedge T(x) && (4)
\end{aligned}
$$



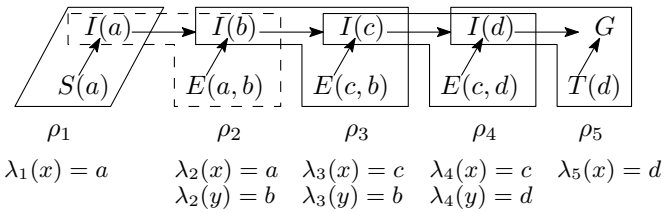


Figure 1 *Top left:* Symmetric Datalog program $\mathcal{P}$. *Top right:* Input structure $\mathbf{G}$ where the binary relation $E^{\mathbf{G}}$ is specified by the digraph. *Bottom:* Visualization of a $\mathcal{P}$-derivation $\mathscr{D}_{\mathcal{P}}(\mathbf{G}) = (\rho_1, \lambda_1), \ldots, (\rho_5, \lambda_5)$ for $\mathbf{G}$, where ρ_1 is nonrecursive, ρ_2, ρ_4 are rules of type 2, ρ_3 is a rule of type 4, and ρ_5 is the goal rule. For example, the dashed box corresponds to rule ρ_2, and it is the rule $I(y) \leftarrow I(x) \wedge E(x,y)$ of $\mathcal{P}$, where λ_2 assigns a to variable x and b to variable y. Observe that $\mathscr{D}_{\mathcal{P}}(\mathbf{G})$ is read-once.

2.3 Path-Decompositions and Derivations

▶ **Definition 3.** [Path-Decomposition] Let $\mathbf{S}$ be a τ-structure. A (j,k)-*path-decomposition* of $\mathbf{S}$ is a sequence $S_0, \ldots, S_{n-1}$ of subsets of A such that

1. For every $(R, (a_1, \ldots, a_{\mathrm{ar}(R)})) \in \tilde{\mathbf{A}}$, $\exists \ell \in \{0, \ldots, n-1\}$ such that $\{a_1, \ldots, a_{\mathrm{ar}(R)}\} \subseteq S_\ell$;
2. If $a \in S_i \cap S_{i'}$ $(i < i')$ then $a \in S_\ell$ for all $i < \ell < i'$;
3. $\forall \ell \in \{0, \ldots, n-1\}$, $|S_\ell| \leq k$, and $\forall \ell \in \{0, \ldots, n-2\}$, $|S_\ell \cap S_{\ell+1}| \leq j$.

For ease of notation, it will be useful to introduce a concept closely related to path-decompositions. Let τ be a vocabulary. Let $\mathbf{S}$ be a τ-structure that can be expressed as $\mathbf{S} = \mathbf{S}_0 \cup \cdots \cup \mathbf{S}_{n-1}$, where the $S_0, \ldots, S_{n-1}$ (the universes of the $\mathbf{S}_i$) satisfy properties 2 and 3 above. Note that $\cup$ here denotes union, *not* disjoint union of τ-structures. We say that $\mathbf{S}$ is a (j,k)-*path*, and that $(\mathbf{S}_0, \ldots, \mathbf{S}_{n-1})$ is a (j,k)-*path representation* of $\mathbf{S}$. We denote (j,k)-path representations with script letters, e.g. $\mathscr{S} = (\mathbf{S}_0, \ldots, \mathbf{S}_{n-1})$. The substructure $\mathbf{S}_i \cup \cdots \cup \mathbf{S}_{i'}$ of $\mathbf{S}$ (assuming a (j,k)-representation is fixed) is denoted with $\mathbf{S}_{[i,j]}$. We call n the *length* of the representation. Obviously, a structure is a (j,k)-path iff it admits a (j,k)-path-decomposition.

Let $\mathscr{D} = (\rho_1, \lambda_1), \ldots, (\rho_q, \lambda_q)$ be a derivation for some linear or symmetric program $\mathcal{P}$ with vocabulary τ. We can extract from $\mathscr{D}$ a τ-structure $\mathrm{Ex}(\mathscr{D})$ such that $\mathscr{D}$ is a derivation for $\mathrm{Ex}(\mathscr{D})$. We specify $\mathrm{Ex}(\mathscr{D})$ as a tuple structure $\tilde{\mathbf{A}}$: for each $R(\mathbf{t})$ that appears in $\mathscr{D}$

$(R \in \tau)$, we add the pair $(R, \mathbf{t})$ to $\tilde{\mathbf{A}}$, and set $\tilde{A}$ to be the set of those elements that appear in a tuple.

Let $\mathscr{D} = (\rho_1, \lambda_1), \ldots, (\rho_q, \lambda_q)$ be a derivation. For each x that is in a rule ρ_ℓ for some $\ell \in [q]$, call x^ℓ the *indexed version of* x. We define an equivalence relation $\mathrm{Eq}(\mathscr{D})$ on the set of indexed variables of $\mathscr{D}$. First we define a graph $G = (V, E)$ as:

- V is the set of all indexed versions of variables in $\mathscr{D}$;
- $(x^\ell, y^{\ell'}) \in E$ if $\ell' = \ell + 1$, x is the i-th variable of the head IDB I of ρ_ℓ, and y is the i-th variable of the body IDB I of $\rho_{\ell+1}$.

Two indexed variables x^ℓ and $y^{\ell'}$ are related in $\mathrm{Eq}(\mathscr{D})$ if they are connected in G. Observe that if $C = \left\{x_1^{\ell_1}, x_2^{\ell_2}, \ldots, x_c^{\ell_c}\right\}$ is a connected component of G, then it must be that $\lambda_{\ell_1}(x_1) = \lambda_{\ell_2}(x_2) = \cdots = \lambda_{\ell_c}(x_c)$.

▶ **Definition 4** (Free Derivation)**.** Let $\mathcal{P}$ be a linear Datalog program and $\mathscr{D} = (\rho_0, \lambda_0), \ldots, (\rho_q, \lambda_q)$ be a derivation for $\mathcal{P}$. Then $\mathscr{D}$ is said to be *free* if for any two $(x^\ell, y^{\ell'}) \notin \mathrm{Eq}(\mathscr{D})$, $\lambda_\ell(x) \neq \lambda_{\ell'}(y)$.

Intuitively, this definition says that $\mathscr{D}$ is free if any two variables in $\mathscr{D}$ which are not "forced" to have the same value are assigned different values.

2.4 Canonical Programs

Fix a τ-structure $\mathbf{B}$ and $j \leq k$. Let $Q_1, \ldots, Q_n$ be all possible at most j-ary relations over B. The *canonical linear* (j,k)*-Datalog program for* $\mathbf{B}$ *(*(j,k)-CanL($\mathbf{B}$)*)* contains an IDB I_m of the same arity as Q_m for each $m \in [n]$. The rule $I_c(\mathbf{x}) \leftarrow I_d(\mathbf{y}) \wedge \bar{E}(\mathbf{z})$ belongs to the canonical program if it contains at most k variables, and the implication $Q_c(\mathbf{x}) \leftarrow Q_d(\mathbf{y}) \wedge \bar{E}(\mathbf{z})$ is true for all possible instantiation of the variables to elements of B. The goal predicate of this program is the 0-ary IDB I_g, where $Q_g = \emptyset$.

The *canonical symmetric* (j,k)*-Datalog program for* $\mathbf{B}$ *(*(j,k)-CanS($\mathbf{B}$)*)* has the same definition as (j,k)-CanL($\mathbf{B}$), except that it has less rules due to the following additional restriction. If $I_c(\mathbf{x}) \leftarrow I_d(\mathbf{y}) \wedge \bar{E}(\mathbf{z})$ is in the program, then both $Q_c(\mathbf{x}) \leftarrow Q_d(\mathbf{y}) \wedge \bar{E}(\mathbf{z})$ and $Q_d(\mathbf{y}) \leftarrow Q_c(\mathbf{x}) \wedge \bar{E}(\mathbf{z})$ must hold for all possible instantiation of the variables to elements of B. The program (j,k)-CanS($\mathbf{B}$) is obviously symmetric. When it is clear from the context, we write CanL($\mathbf{B}$) and CanS($\mathbf{B}$) instead of (j,k)-CanL($\mathbf{B}$) and (j,k)-CanS($\mathbf{B}$), respectively.

2.5 Defining CSPs

The following discussion applies not just to Datalog but also to its symmetric and linear fragments. It is easy to see that the class of structures accepted by a Datalog program is homomorphism-closed, and therefore it is not possible to define CSP($\mathbf{B}$) in Datalog. However, it is often possible to define co-CSP($\mathbf{B}$) in Datalog. The following definition is key.

▶ **Definition 5** (Obstruction Set)**.** A set $\mathcal{O}$ of τ-structures is called an *obstruction set* for $\mathbf{B}$, if for any τ-structure $\mathbf{A}$, $\mathbf{A} \not\to \mathbf{B}$ iff there exists $\mathbf{S} \in \mathcal{O}$ such that $\mathbf{S} \to \mathbf{A}$.

If $\mathcal{O}$ above can be chosen to have property X, then we say that $\mathbf{B}$ has X-duality.

3 On CSPs in symmetric Datalog

3.1 Definitions

An *oriented path* is a digraph obtained by orienting the edges of an undirected path, i.e. an oriented path has vertices $v_0, \ldots, v_{q+1}$ and edges $e_0, \ldots, e_q$, where e_i is either (v_i, v_{i+1}),

or (v_{i+1}, v_i). The *length* of an oriented path is the number of edges it contains. We call (v_i, v_{i+1}) a *forward edge* and (v_{i+1}, v_i) a *backward edge*. Oriented paths can be thought of as relational structures over the vocabulary $\{E^2\}$, so we denote them with boldface letters.

For an oriented path $\mathbf{P}$, we can find a mapping level : $P \to \{0, 1, 2, \dots\}$ such that $\text{level}(b) = \text{level}(a) + 1$ whenever (a, b) is an edge of $\mathbf{P}$. Clearly, there is a unique such mapping with the smallest possible values. The *level of an edge* (a, b) of $\mathbf{P}$ is $\text{level}(a)$, i.e. the level of the starting vertex of (a, b). The $\text{height}(\mathbf{P})$ of an oriented path $\mathbf{P}$ is $\max_{a \in P} \text{level}(a)$. We say that an oriented path $\mathbf{P}$ is *minimal* if there is precisely one vertex a such that $\text{level}(a) = 0$, and precisely one vertex b such that $\text{level}(b) = \text{height}(\mathbf{P})$.

A *zigzag operator* ξ takes a (j, k)-path representation $\mathscr{S} = (\mathbf{S}_0, \dots, \mathbf{S}_{n-1})$ of a (j, k)-path $\mathbf{S}$ and a minimal oriented path $\mathbf{P} = e_0, \dots, e_q$ such that $\text{height}(\mathbf{P}) = n$, and it returns another (j, k)-path $\xi(\mathscr{S}, \mathbf{P})$. Intuitively, $\xi(\mathscr{S}, \mathbf{P})$ is the (j, k)-path $\mathbf{S}$ "modulated" by $\mathbf{P}$ such that the forward and backward edges e_i of $\mathbf{P}$ are mimicked in $\xi(\mathscr{S}, \mathbf{P})$ by "forward and backward" copies of $\mathbf{S}_{\text{level}(e_i)}$. Before the formal definition, it could help the reader to look at the right side of Fig. 2, where the oriented path used to modulate the (j, k)-path over the vocabulary E^2 (i.e. digraphs) with representation $(\mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_2)$ is $\mathbf{P}$ on the left side. The right side is a more abstract example, and the reader might find it useful after reading the definition.

We inductively define the (j, k)-path $\xi(\mathscr{S}, \mathbf{P})$ as $(\mathbf{S}_{e_0}, \mathbf{S}_{e_1}, \dots, \mathbf{S}_{e_q})$ together with a sequence of isomorphisms $\varphi_{e_0}, \varphi_{e_1}, \dots, \varphi_{e_q}$, where φ_{e_i} is an isomorphism from $\mathbf{S}_{e_i}$ to $\mathbf{S}_{\text{level}(e_i)}$, $0 \leq i \leq q$. For the base case, we define $\mathbf{S}_{e_0}$ to be an isomorphic copy of $\mathbf{S}_0$, and φ_{e_0} to be the isomorphism that maps $\mathbf{S}_{e_0}$ back to $\mathbf{S}_0$. Assume inductively that $\mathbf{S}_{e_0}, \dots, \mathbf{S}_{e_{i-1}}$ and $\varphi_{e_0}, \dots, \varphi_{e_{i-1}}$ are already defined. Let $\mathbf{S}'_{e_i}$ be an isomorphic copy of $\mathbf{S}_{\text{level}(e_i)}$ with domain disjoint from $S_{e_0} \cup \dots \cup S_{e_{i-1}}$, and fix φ'_{e_i} to be the isomorphism that maps back S'_{e_i} to $S_{\text{level}(e_i)}$. We "glue" $\mathbf{S}'_{e_i}$ to $\mathbf{S}_{e_{i-1}}$ by renaming some elements of $\mathbf{S}'_{e_i}$ to elements of $\mathbf{S}_{e_{i-1}}$. To facilitate understanding, we can think of the already constructed structures $\mathbf{S}_{e_0}, \dots, \mathbf{S}_{e_{i-1}}$ as labels of the edges $e_0, \dots, e_{i-1}$ of $\mathbf{P}$, respectively, and we want to determine $\mathbf{S}_{e_i}$, the label of the next edge. The connection between $\mathbf{S}_{e_{i-1}}$ and $\mathbf{S}_{e_i}$ will be defined such that $\mathbf{S}_{e_{i-1}}$ and $\mathbf{S}_{e_i}$ "mimic" the orientation of the edges e_{i-1} and e_i.

We resume our formal definition. Set $\ell = \text{level}(e_i)$, and let $\ell' = \ell - 1$ if e_i is a forward edge, and $\ell' = \ell + 1$ if e_i is a backward edge. If an element $x \in S'_{e_i}$ and an element $y \in S_{e_{i-1}}$ are both copies of the same element $a \in S_\ell \cap S_{\ell'}$, then rename x to y in S'_{e_i}. After all such elements are renamed, $\mathbf{S}'_{e_i}$ becomes $\mathbf{S}_{e_i}$. That is, for all $a \in S_\ell \cap S_{\ell'}$, rename $\varphi'^{-1}_{e_i}(a)$ in $\mathbf{S}'_{e_i}$ to $\varphi^{-1}_{e_{i-1}}(a)$ to obtain $\mathbf{S}_{e_i}$.

We define the isomorphism φ_{e_i} from $\mathbf{S}_{e_i}$ to $\mathbf{S}_{\text{level}(e_i)}$ as:

$$\varphi_{e_i}(x) = \begin{cases} \varphi'_{e_i}(x) & \text{if } x \in S_{e_i} \text{ and } x \notin S_{e_{i-1}} \\ \varphi_{e_{i-1}}(x) & \text{if } x \in S_{e_i} \cap S_{e_{i-1}}. \end{cases}$$

3.2 Two Dualities for Symmetric Datalog

The two main theorems (Theorems 9 and 16) of this section can be combined to obtain:

▶ **Theorem 6.** *For a finite structure* $\mathbf{B}$*, TFAE:*

1. *There is a symmetric Datalog program that defines* co-CSP($\mathbf{B}$)*;*
2. $\mathbf{B}$ *has symmetric bounded pathwidth duality (for some parameters);*
3. $\mathbf{B}$ *has piecewise symmetric bounded pathwidth duality (for some parameters).*

Details follow.

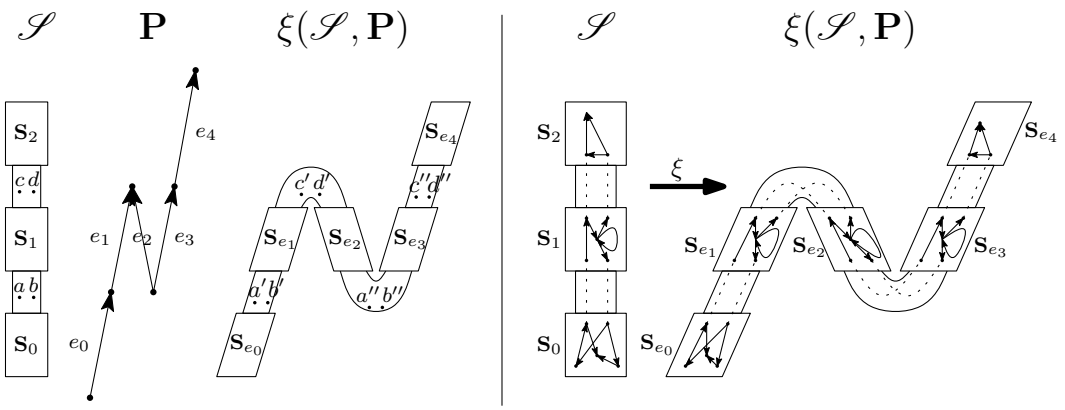


Figure 2 *Left:* Applying a zigzag operator to the (j,k)-path $\mathbf{S}$ with the (j,k)-representation $\mathscr{S} = (\mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_2)$. Suppose that $S_0 \cap S_1 = \{a, b\}$ and $S_1 \cap S_2 = \{c, d\}$. We demonstrate how $\mathbf{S}_{e_0}$ and $\mathbf{S}_{e_2}$ are obtained. $\mathbf{S}_{e_0}$ is a disjoint copy of $\mathbf{S}_0$ (and the copy of a and b in $\mathbf{S}_{e_0}$ are a' and b', respectively). To obtain $\mathbf{S}_{e_2}$, first make a disjoint copy $\mathbf{S}'_{e_2}$ of $\mathbf{S}_{\text{level}(e_2)} = \mathbf{S}_1$. Set $\ell = \text{level}(e_2) = 1$. Since e_1 is a forward edge and e_2 is a backward edge, $\ell' = \ell + 1 = 2$. Therefore to "glue" $\mathbf{S}'_{e_2}$ to $\mathbf{S}_{e_1}$, we need to look at $S_\ell \cap S_{\ell'} = \{c, d\}$. Assume that the copy of c and d in $\mathbf{S}_{e_1}$ are c' and d', respectively. Furthermore, assume that the copy of c and d in $\mathbf{S}'_{e_2}$ are $\tilde{c}$ and $\tilde{d}$, respectively. To obtain $\mathbf{S}_{e_2}$, we rename $\tilde{c}$ to c', and $\tilde{d}$ to d' in $\mathbf{S}'_{e_2}$. *Right:* A specific example when $\mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_2$ are the digraphs in the boxes. The dashed lines indicate identification of vertices.

3.2.1 Symmetric Bounded Pathwidth Duality

▶ **Definition 7** ((j,k)-symmetric)**.** Assume that $\mathcal{O}$ is a set of (j,k)-paths. Suppose furthermore that a (j,k)-path representation can be fixed for each structure in $\mathcal{O}$ such that the following holds. For every $\mathbf{S} \in \mathcal{O}$ with representation $\mathscr{S}$ of some length n, and every minimal oriented path $\mathbf{P}$ of height n, it holds that $\xi(\mathscr{S}, \mathbf{P}) \in \mathcal{O}$. Then $\mathcal{O}$ is said to be (j,k)-*symmetric*.

▶ **Definition 8** (SBPD)**.** A structure $\mathbf{B}$ has (j,k)-*symmetric bounded pathwidth duality* ((j,k)-*SBPD*) if there is an obstruction set $\mathcal{O}$ for $\mathbf{B}$ that consists of (j,k)-paths, and in addition, $\mathcal{O}$ is (j,k)-symmetric.

▶ **Theorem 9.** *For a finite structure* $\mathbf{B}$, co-CSP($\mathbf{B}$) *can be defined by a symmetric* (j,k)*-Datalog program if and only if* $\mathbf{B}$ *has* (j,k)*-SBPD.*

To prove Theorem 9, first we prove Lemma 10 using the standard canonical Datalog argument:

▶ **Lemma 10.** *If* CanS($\mathbf{B}$) *accepts a structure* $\mathbf{A}$*, then* $\mathbf{A} \not\to \mathbf{B}$.

The following is the main technical lemma of the section.

▶ **Lemma 11.** *For any* τ*-structures* $\mathbf{A}$ *and* $\mathbf{B}$*, if there exists a structure* $\mathbf{S}$ *with a* (j,k)*-path representation* $\mathscr{S}$ *of some length* n *such that* $\mathbf{S} \to \mathbf{A}$*, and for any minimal oriented path* $\mathbf{P}$ *of height* n*, it holds that* $\xi(\mathscr{S}, \mathbf{P}) \not\to \mathbf{B}$*, then* (j,k)-CanS($\mathbf{B}$) *accepts* $\mathbf{A}$.

Proof of Theorem 9. If CSP($\mathbf{B}$) is defined by a symmetric (j,k)-Datalog program $\mathcal{P}$, then using the symmetric property of $\mathcal{P}$, it is laborious but straightforward to show that

$$\mathcal{O} = \bigcup_{\substack{\mathscr{D} \text{ is a free} \\ \text{derivation of } \mathcal{P}}} \{\text{Ex}(\mathscr{D})\}$$

is a (j,k)-symmetric obstruction set for $\mathbf{B}$.

For the converse, assume that $\mathbf{B}$ has (j,k)-SBPD. Let $\mathcal{O}$ be a symmetric obstruction set of width (j,k) for $\mathbf{B}$. We claim that (j,k)-CanS($\mathbf{B}$) defines CSP($\mathbf{B}$). Assume that $\mathbf{A} \to \mathbf{B}$. Then by Lemma 10, (j,k)-CanS($\mathbf{B}$) does not accept $\mathbf{A}$. Suppose now that $\mathbf{A} \not\to \mathbf{B}$. Then by assumption, there exists a (j,k)-path $\mathbf{S} \in \mathcal{O}$ with a representation $\mathscr{S}$ of length n such that $\mathbf{S} \to \mathbf{A}$. Furthermore, since $\mathcal{O}$ is symmetric, for any minimal oriented path $\mathbf{P}$ of height n, $\xi(\mathscr{S}, \mathbf{P}) \not\to \mathbf{B}$. It follows from Lemma 11 that CanS($\mathbf{B}$) accepts $\mathbf{A}$. ◀

From the above proof it is obvious that:

▶ **Corollary 12** ([8]). *If a symmetric (j,k)-Datalog program defines* CSP($\mathbf{B}$)*, then so does* (j,k)-CanS($\mathbf{B}$).

3.2.2 Piecewise Symmetric Bounded Pathwidth Duality

Piecewise symmetric bounded pathwidth duality (PSBPD) for symmetric Datalog is less stringent than SBPD; however, the price is larger program width. Although the following definitions might seem technical, the general idea is simple: a piecewise symmetric obstruction set $\mathcal{O}$ does not need to contain all (j,k)-paths obtained by zigzagging (j,k)-paths in $\mathcal{O}$ in all possible ways. It is sufficient to zigzag a (j,k)-path $\mathbf{S}$ using only oriented paths which "avoid" certain segments of $\mathbf{S}$: some constants c and d are fixed for $\mathcal{O}$, and there are at most c fixed segments of $\mathbf{S}$ that are avoided by the zigzag operator, each of size at most d. We give the formal definitions.

▶ **Definition 13** ((c,d)-filter). Let $\mathbf{S}$ be a (j,k)-path with a representation $\mathscr{S} = \mathbf{S}_0, \ldots, \mathbf{S}_{n-1}$. A (c,d)-*filter* $\mathscr{F}$ for $\mathscr{S}$ is a set of intervals $\{[s_1,t_1],[s_2,t_2],\ldots,[s_{c'},t_{c'}]\}$ such that

- $c' \leq c$; $0 \leq s_1$; $t_{c'} \leq n-1$; $s_i \leq t_i, \forall i \in [c']$; and $t_\ell + 2 \leq s_{\ell+1}, \forall \ell \in [c'-1]$;
- $|\bigcup_{i \in [s_\ell, t_\ell]} S_i| \leq d, \forall \ell \in [c']$.

Elements of $\mathscr{F}$ are called *delimiters*. An oriented path $\mathbf{P}$ of height n *obeys* a (c,d)-filter $\mathscr{F}$ if for any delimiter $[s_i,t_i] \in \mathscr{F}$, the set of edges e of $\mathbf{P}$ such that $s_i \leq \text{level}(e) \leq t_i$ form a (single) directed path. A demonstration is given in Fig. 3.

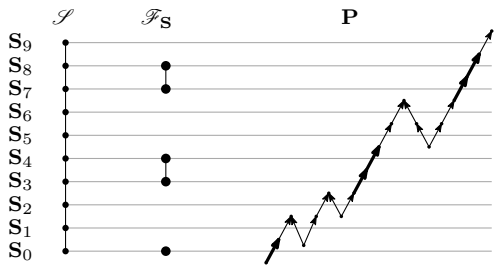


Figure 3 $\mathscr{S}$ is a (j,k)-path representation of $\mathbf{S}$. $\mathscr{F}_\mathbf{S}$ is the $(3,2k)$-filter $\{[0,0][3,4][7,8]\}$ for $\mathscr{S}$. $\mathbf{P}$ is an oriented path that obeys the filter. For example, observe that the edges at levels 3 and 4 form a directed subpath, and that "zigzagging" happens only at those parts of $\mathbf{P}$ that do not fall into the intervals of the filter.

▶ **Definition 14** (Piecewise Symmetric). Assume that $\mathcal{O}$ is a set of (j,k)-paths, and c and d are nonnegative integers. Suppose furthermore that for each $\mathbf{S} \in \mathcal{O}$, there is a (j,k)-path representation $\mathscr{S}$, and a (c,d)-filter $\mathscr{F}_\mathbf{S}$ such that the following holds. For every $\mathbf{S} \in \mathcal{O}$ of some length n, and every minimal oriented path $\mathbf{P}$ of height n that obeys the filter $\mathscr{F}_\mathbf{S}$, it holds that $\xi(\mathscr{S}, \mathbf{P}) \in \mathcal{O}$. Then $\mathcal{O}$ is (j,k,c,d)-*piecewise symmetric*.

Roughly speaking, an oriented path $\mathbf{P}$ is allowed to modulate only those segments of $\mathscr{S}$ which do not correspond to any delimiters in $\mathscr{F}_{\mathbf{S}}$. Compare Definition 14 with Definition 7, and observe that the only difference is that in the piecewise case, the oriented paths must be of a restricted form. Therefore a set that is (j,k)-symmetric is also (j,k,c,d)-piecewise symmetric for any c and d. We simply associate the empty (c,d)-filter with each structure.

▶ **Definition 15** (PSBPD). A structure $\mathbf{B}$ has *(j,k,c,d)-piecewise symmetric bounded pathwidth duality ((j,k,c,d)-PSBPD)* if there is an obstruction set $\mathcal{O}$ for $\mathbf{B}$ that consists of (j,k)-paths, and in addition, $\mathcal{O}$ is (j,k,c,d)-piecewise symmetric.

▶ **Theorem 16.** *For a finite structure $\mathbf{B}$, $\mathbf{B}$ has SBPD (for some parameters) if and only if $\mathbf{B}$ has PSBPD (for some parameters).*

3.3 Applications

3.3.1 Datalog + Maltsev ⇒ symmetric Datalog

Using SBPD, we give a short and simple re-proof of the main result of [8]:

▶ **Theorem 17** ([8]). *Let $\mathbf{B}$ be a finite core structure. If $\mathbf{B}$ is invariant under a Maltsev operation and* co-CSP($\mathbf{B}$) *is definable in Datalog, then* co-CSP($\mathbf{B}$) *is definable in symmetric Datalog (and therefore* CSP($\mathbf{B}$) *is in* L *by [10]).*

We only need to show that if co-CSP($\mathbf{B}$) is in linear Datalog and $\mathbf{B}$ is preserved by a Maltsev operation, then co-CSP($\mathbf{B}$) is in symmetric Datalog. The "jump" from Datalog to linear Datalog essentially follows from already established results, as observed in [8]. Therefore to re-prove Theorem 17, we show the following lemma using an SBPD argument.

▶ **Lemma 18.** *If* co-CSP($\mathbf{B}$) *is definable by a linear Datalog program and $\mathbf{B}$ is invariant under a Maltsev operation m, then* co-CSP($\mathbf{B}$) *is definable by a symmetric Datalog program.*

To get ready for the proof of Lemma 18, we define an N of size s as an oriented path that consists of s forward edges, followed by s backward edges, followed by another s forward edges. Proposition 19 is easy to prove, and the Maltsev properties are used in Lemma 20.

▶ **Proposition 19.** *A minimal oriented path is either a directed path, or it contains a subpath which is an N.*

▶ **Lemma 20.** *Let $\mathbf{B}$ be a structure invariant under a Maltsev operation m, $\mathbf{S}$ be a (j,k)-path with a (j,k)-representation $\mathscr{S} = (\mathbf{S}_0,\dots,\mathbf{S}_{n-1})$, and $\mathbf{P} = e_0,\dots,e_q$ be a minimal oriented path of height n. If $\xi(\mathscr{S},\mathbf{P}) \to \mathbf{B}$, then $\mathbf{S} \to \mathbf{B}$.*

Proof. Using Proposition 19, there is an index t such that $\mathbf{Q} = e_t, e_{t+1},\dots,e_{t+(3s-1)}$ is an N of size s in $\mathbf{P}$. Assume that the first and last vertices of $\mathbf{Q}$ are v and w, respectively. Let $\mathbf{P}'$ be the oriented path obtained from $\mathbf{P}$ by removing $\mathbf{Q}$, and adding a directed path $\mathbf{Q}' = f_t, f_{t+1},\dots,f_{t+(s-1)}$ of length s from v to w. We claim that there is a homomorphism γ from $\xi(\mathscr{S},\mathbf{P}')$ to $\mathbf{B}$. Once this is established, a repetition of this argument sufficiently many times yields that $\mathbf{S} \to \mathbf{B}$ because if we repeatedly "remove" N-s from a minimal oriented path, eventually we must reach a directed path.

Let $\xi(\mathscr{S},\mathbf{P}) = (\mathbf{S}_{e_0},\dots,\mathbf{S}_{e_q})$, and $\varphi_{e_0},\dots,\varphi_{e_q}$ be the corresponding isomorphisms (recall the zigzag operator definition in Section 3.1). Similarly, let $\xi(\mathscr{S},\mathbf{P}') = (\mathbf{S}_{f_0},\dots,\mathbf{S}_{f_{q-2s}})$, and $\psi_{f_0},\dots,\psi_{f_{q-2s}}$ be the corresponding isomorphisms. Because $\mathbf{S}_{[e_0,e_{t-1}]}$ and $\mathbf{S}_{[e_{t+3s},e_q]}$ are isomorphic to $\mathbf{S}_{[f_0,f_{t-1}]}$ and $\mathbf{S}_{[f_{t+s},f_{q-2s}]}$, respectively, γ for elements in $S_{[f_0,f_{t-1}]} \cup S_{[f_{t+s},e_{q-2s}]}$ is defined in the natural way. It remains to define γ for every $d \in S_{[f_t,f_{t+(s-1)}]}$.

Assume that $d \in S_{f_{t+\ell}}$ for some $\ell \in \{0, \dots, s-1\}$. Find the original of d in $\mathbf{S}$ and let it be d_o, i.e. $d_o = \psi_{f_{t+\ell}}(d)$. Then we find the three copies d_1, d_2, d_3 of d_o in $\mathbf{S}_{[f_t, f_{t+(3s-1)}]}$. That is, first we find the three edges $e_{\ell_1}, e_{\ell_2}, e_{\ell_3}$ of $\mathbf{Q}$ which have the same level as $f_{t+\ell}$ (all levels are with respect to $\mathbf{P}$ and $\mathbf{P}'$). Then $d_i = \varphi^{-1}_{e_{\ell_i}}(d_o)$, $i \in [3]$. We define $\gamma(d) = m(d_1, d_2, d_3)$. By the Maltsev properties of m, γ is well-defined. As $\mathbf{B}$ is invariant under m, $\xi(\mathscr{S}, \mathbf{P}') \xrightarrow{\gamma} \mathbf{B}$. ◀

Proof of Lemma 18. If co-CSP($\mathbf{B}$) can be defined by a linear (j,k)-Datalog program, then there is an obstruction set $\mathcal{O}$ for $\mathbf{B}$ in which every structure is a (j,k)-path by [6]. We construct a symmetric obstruction set $\mathcal{O}_{sym}$ for $\mathbf{B}$ as follows. First we define a sequence of sets $\mathcal{O}_1, \mathcal{O}_2, \dots$ inductively, where $\mathcal{O}_1 = \mathcal{O}$. To construct $\mathcal{O}_{i+1}$, for every (j,k)-path $\mathbf{S}$ with a (j,k)-representation $\mathscr{S} = \mathbf{S}_0, \dots, \mathbf{S}_{n-1}$ in $\mathcal{O}_i$, and any minimal oriented path $\mathbf{P}$ of height n, place $\xi(\mathscr{S}, \mathbf{P})$ into $\mathcal{O}_{i+1}$. Set $\mathcal{O}_{sym} = \bigcup_{1 \le i} \mathcal{O}_i$. By construction $\mathcal{O}_{sym}$ is symmetric.

Observe that $\mathcal{O} \subseteq \mathcal{O}_{sym}$, so it remains to show that no element of $\mathcal{O}_{sym}$ maps to $\mathbf{B}$. For contradiction, take an element $\mathbf{T} \in \mathcal{O}_{sym}$ such that $\mathbf{T} \to \mathbf{B}$. By definition of $\mathcal{O}_{sym}$, there is a $\mathbf{T}_0 \in \mathcal{O}$ and a sequence of oriented paths $\mathbf{P}_1, \dots, \mathbf{P}_d$ such that $\mathbf{T}$ is obtained from $\mathbf{T}_0$ as follows. First $\mathbf{T}_1 = \xi(\mathscr{T}_0, \mathbf{P}_1)$ was constructed (and placed into $\mathcal{O}_1$), where $\mathscr{T}_0$ is a (j,k)-path representation of $\mathbf{T}_0$. Then $\mathbf{T}_2 = \xi(\mathscr{T}_1, \mathbf{P}_2)$ was constructed (and placed into $\mathcal{O}_2$), where $\mathscr{T}_1$ is a (j,k)-path representation of $\mathbf{T}_1$, and so on, until $\mathbf{T}_d = \mathbf{T}$ is constructed. We find the largest i such that $\mathbf{T}_i \not\to \mathbf{B}$. Lemma 20 tells us that if $\mathbf{T}_{i+1} \to \mathbf{B}$, then $\mathbf{T}_i \to \mathbf{B}$, a contradiction. ◀

3.3.2 A class of oriented paths for which the CSP is in L, and a class for which the CSP is NL-complete

In this section we define a class $\mathcal{C}$ of oriented paths such that if $\mathbf{B} \in \mathcal{C}$ then co-CSP($\mathbf{B}$) is in symmetric Datalog. Our strategy is to find an obstruction set $\mathcal{O}$ for $\mathbf{B} \in \mathcal{C}$, and then to show that our obstruction set is piecewise symmetric. We need some notation.

We say that a directed path is *forward* to mean that its first and last vertices are the vertices with indegree zero and outdegree zero, respectively. Let $\mathbf{P}$ be an oriented path with first vertex v and last vertex w. Then the *reverse* of $\mathbf{P}$, denoted with $\bar{\mathbf{P}}$, is a copy of the oriented path $\mathbf{P}$ in the reverse direction, i.e. the first vertex of $\bar{\mathbf{P}}$ is a copy of w and its last vertex is a copy of v. Let $\mathbf{Q}$ be another oriented path. The *concatenation* of $\mathbf{P}$ and $\mathbf{Q}$ is the oriented path $\mathbf{PQ}$ in which the last vertex of $\mathbf{P}$ is identified with the first vertex of $\mathbf{Q}$. For a nonnegative integer r, $\mathbf{P}^r$ denotes $\mathbf{P}_1\mathbf{P}_2 \dots \mathbf{P}_r$, where the $\mathbf{P}_\ell$ are disjoint copies of $\mathbf{P}$. Given two vertices v and w, we denote the presence of an edge from v to w with $v \to w$.

▶ **Definition 21** (Wave)**.** If an oriented path $\mathbf{Q}$ can be expressed as $\mathbf{E}_1(\mathbf{P}\bar{\mathbf{P}})^r\mathbf{P}\mathbf{E}_2$, where $\mathbf{E}_i$ ($i \in [2]$) denotes the forward directed path that is a single edge, $\mathbf{P}$ is a forward directed path of length ℓ, and $r \ge 0$, then $\mathbf{Q}$ is called an *ℓ-wave*. A 2-wave is shown in Fig. 4, *1.*

▶ **Theorem 22.** *Let* $\mathbf{Q}$ *be a wave. Then* $\mathbf{Q}$ *has PSBPD,* co-CSP($\mathbf{Q}$) *is definable in symmetric Datalog, and* CSP($\mathbf{Q}$) *is in* L.

We state the following generalization of waves.

▶ **Definition 23** (Staircase)**.** A *monotone wave* is an oriented path of the form $(\bar{\mathbf{P}}\mathbf{P})^r\bar{\mathbf{P}}$, where $\mathbf{P}$ is a forward directed path and $r \ge 0$. We call the vertices of a monotone wave in the topmost level *peaks*, and the vertices in the bottommost level *troughs*.

If a minimal oriented path $\mathbf{Q}$ can be expressed as $\mathbf{P}_1\mathbf{W}_1\mathbf{P}_2\mathbf{W}_2 \dots \mathbf{P}_{n-1}\mathbf{W}_{n-1}\mathbf{P}_n$, where $\mathbf{P}_1, \dots, \mathbf{P}_n$ are forward directed paths, $\mathbf{W}_1, \dots, \mathbf{W}_{n-1}$ are monotone waves, and for any $i \in [n-1]$, the troughs of $\mathbf{W}_i$ are in a level strictly below the level of the troughs of $\mathbf{W}_{i+1}$,

and also, the peaks of $\mathbf{W}_i$ are in a level strictly below the level of the peaks of $\mathbf{W}_{i+1}$, then $\mathbf{Q}$ is called a *staircase*. An example is given in Fig. 4, *2*.

▶ **Theorem 24.** *Let* $\mathbf{Q}$ *be a staircase. Then* $\mathbf{Q}$ *has PSBPD,* co-CSP($\mathbf{Q}$) *is definable in symmetric Datalog, and* CSP($\mathbf{Q}$) *is in* L.

We also give a large class of oriented paths for which the CSP is NL-complete.

▶ **Theorem 25.** *Let* $\mathbf{B}$ *be a core oriented path that contains a subpath* $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ *of some height* h *with the following properties:* $\mathbf{P}_1, \mathbf{P}_2$ *and* $\mathbf{P}_3$ *are minimal oriented paths, they all have height* h*, and there is a minimal oriented path* $\mathbf{Q}$ *of height* h *such that* $\mathbf{Q} \to \mathbf{P}_1$, $\mathbf{Q} \to \mathbf{P}_3$ *but* $\mathbf{Q} \not\to \mathbf{P}_2$*. Then* CSP($\mathbf{B}$) *is* NL*-complete.*

An example is given in Fig. 4, *3* and *4*.

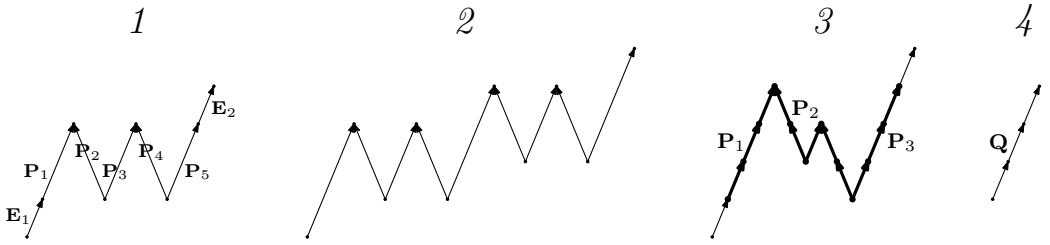


Figure 4 *1:* A 2-wave. *2:* A staircase. *3:* An example oriented path for which the CSP is NL-complete. *4:* The oriented path $\mathbf{Q}$ in Theorem 25 corresponding to the oriented path in *3*.

4 On CSPs in NL

4.1 Preliminaries and Definitions

Let τ be a vocabulary. A *successor* τ*-structure* $\mathbf{S}$ is a relational structure with vocabulary $\tau \cup \{\mathfrak{first}, \mathfrak{last}, \mathfrak{suc}\}$, where $\mathfrak{first}$ and $\mathfrak{last}$ are unary symbols and $\mathfrak{suc}$ is a binary symbol. The domain S is defined as $\{1, \ldots, n\}$, $\mathfrak{first}^{\mathbf{S}} = \{1\}$, $\mathfrak{last}^{\mathbf{S}} = \{n\}$, and $\mathfrak{suc}^{\mathbf{S}}$ contains all pairs $(i, i+1)$, $i \in [n-1]$. Because $\mathfrak{first}^{\mathbf{S}}, \mathfrak{last}^{\mathbf{S}}$ and $\mathfrak{suc}^{\mathbf{S}}$ depend only on n, they are called *built-in* relations. When we say that a class of successor structures is homomorphism/isomorphism-closed, all structures under consideration are successor structures, and we understand that homomorphism/isomorphism closure, respectively, is required only for non-built-in relations.

▶ **Definition 26** (Split Operation)**.** A *split operation* produces a τ-structure $\mathbf{A}'$ from a τ-structure $\mathbf{A}$ as follows. For an element $a \in A$ let T_a be defined as

$$T_a = \left\{(\mathbf{t}, R, i) \mid \mathbf{t} = (t_1, \ldots, t_r) \in R^{\mathbf{A}} \text{ where } R \in \tau, \text{ and } t_i = a\right\}.$$

If $|T_a| \leq 1$, no split operation can be applied. Otherwise we choose a strict nonempty subset T of T_a, and for each triple $(\mathbf{t}, R, i) \in T$, we replace $\mathbf{t} = (t_1, \ldots, t_r)$ in $R^{\mathbf{A}}$ with $(t_1, \ldots, t_{i-1}, a', t_{i+1}, \ldots, t_r)$ to obtain $\mathbf{A}'$ (and $A' = A \cup \{a'\}$).

▶ **Definition 27** (Split-Minimal, Critical)**.** Let $\mathcal{C}$ be a class of structures over the same vocabulary. We say that a structure $\mathbf{A} \in \mathcal{C}$ is *split-minimal in* $\mathcal{C}$ if for every possible nonempty sequence of split operations applied to $\mathbf{A}$, the resulting structure is not in $\mathcal{C}$. We say that a structure $\mathbf{A} \in \mathcal{C}$ is *critical in* $\mathcal{C}$ if no proper substructure of $\mathbf{A}$ is in $\mathcal{C}$.

For a class of isomorphism-closed successor τ-structures, criticality and split-minimality is meant only with respect to the non-built-in relations.

▶ **Definition 28** (Read-Once Datalog)**.** Let $\mathcal{P}$ be a (linear, symmetric) Datalog program that defines a class of structures $\mathcal{C}$. If for every critical and split-minimal element of $\mathcal{C}$ there is a $\mathcal{P}$-derivation that is read-once, then we say that $\mathcal{P}$ is *read-once*.

▶ **Definition 29** (Read-Once mnBP1)**.** A *monotone nondeterministic branching program* (mnBP) H with variables $X = \{x_1, \ldots, x_n\}$ computes a Boolean function $f_H : \{0,1\}^n \to \{0,1\}$. H is a directed graph with distinguished nodes s and t and some arcs labeled with variables from X (not all arcs must be labeled). An assignment σ to the variables in X defines in a natural way a subgraph H_σ of H. The function f_H is defined as $f_H(\sigma) = 1$ iff H_σ has a directed path from s to t (an *accepting path*). The size of an mnBP is $|V_H|$.

Let $\mathcal{F}$ be a poly-size family of mnBP1s (mnBP1(poly)) that defines a class of structures $\mathcal{C}$ over a vocabulary τ. (The encoding is done in the straightforward manner, i.e. there is a variable for every possible $(R, \mathbf{t})$ where $R \in \tau$ and $\mathbf{t}$ is a tuple.) If for every structure in $\mathcal{C}$ there is an accepting path that queries every variable at most once, then we say that $\mathcal{F}$ is *read-once*. (This read-once condition can be made a bit weaker.)

We give some examples of problems definable by a 1-linDat($\mathfrak{suc}$) program or by an mnBP1(poly). The program in Section 2.2, Fig. 1 without rule 3 is a read-once linear Datalog($\mathfrak{suc}$) program that defines the problem directed *st*-Conn. To see that this program $\mathcal{P}_{st-\text{Conn}}$ is read-once, let $\mathbf{G}$ be any input that is accepted (we do not even need $\mathbf{G}$ to be critical and split-minimal). Then we find a directed path in $E^{\mathbf{G}}$ connecting an element of $S^{\mathbf{G}}$ to an element of $T^{\mathbf{G}}$ without repeated edges. We build a $\mathcal{P}_{st-\text{Conn}}$-derivation for this path in the obvious way.

Let EvenCliques be the class of undirected graphs which are cliques of even size. A bit of work shows that EvenCliques can be defined with a 1-linDat($\mathfrak{suc}$) program. In fact, we can easily test much more complicated arithmetic properties than the property of being even (e.g. being a power of k) with a 1-linDat($\mathfrak{suc}$) program. We note that EvenCliques or "cliques with any domain size property" cannot be defined by a linear Datalog program because a (nontrivial) set of cliques is never closed under homomorphisms. Since a 1-linDat($\mathfrak{suc}$) program can be converted into an mnBP1(poly), the aforementioned problems can also be defined with an mnBP1(poly).

4.2 Main Results

We simply state the results for 1-linDat($\mathfrak{suc}$) and poly-size families of mnBP1s discussed in the Introduction.

▶ **Theorem 30.** *Let $\mathcal{C}$ be a homomorphism-closed class of successor τ-structures. If $\mathcal{C}$ can be defined by a 1-linDat($\mathfrak{suc}$) program of width (j,k), then every critical and split-minimal element of $\mathcal{C}$ has a $(j, k+j)$-path-decomposition.*

▶ **Corollary 31.** *If* co-CSP($\mathbf{B}$) *can be defined by a 1-linDat($\mathfrak{suc}$) program of width (j,k), then* co-CSP($\mathbf{B}$) *can also be defined by a linear $(j, k+j)$-Datalog program.*

▶ **Theorem 32.** *Let $\mathcal{C}$ be a homomorphism-closed class of successor τ-structures. If $\mathcal{C}$ can be defined by a family of mnBP1s of size $O(n^j)$, then every critical and split-minimal element of $\mathcal{C}$ has a $(j, r+j)$-path-decomposition, where r is the maximum arity of the symbols in τ.*

▶ **Corollary 33.** *If* co-CSP($\mathbf{B}$) *can be defined by a family of mnBP1s of size $O(n^j)$, then* co-CSP($\mathbf{B}$) *can also be defined by a linear $(j, r+j)$-Datalog program, where r is the maximum arity of the relation symbols in the vocabulary of $\mathbf{B}$.*

As discussed before, a wide class of CSPs–CSPs whose associated *variety admits the unary, affine or semilattice types*–does not have bounded pathwidth duality [16]. It follows that all these CSPs are not definable by any 1-linDat($\mathfrak{suc}$) program, or with any mnBP1 of poly-size. An example of such a CSP is the P-complete CSP Horn-3Sat.

References

1. F. Afrati and S. S. Cosmadakis. Expressiveness of restricted recursive queries. In *Proceedings of STOC*, pages 113–126, 1989.
2. Eric Allender, Michael Bauland, Neil Immerman, Henning Schnoor, and Heribert Vollmer. The complexity of satisfiability problems: Refining Schaefer's theorem. *J. Comput. Syst. Sci.*, 75(4):245–254, 2009.
3. L. Barto and M. Kozik. Constraint satisfaction problems of bounded width. In *Proceedings of The 50th Annual Symposium on Foundations of Computer Science (FOCS)*, 2009.
4. A. Bulatov, A. Krokhin, and B. Larose. Dualities for constraint satisfaction problems. In *LNCS Surveys on Complexity of Constraints*, volume 5250, pages 93–124. 2008.
5. C. Carvalho, L. Egri, M. Jackson, and T. Niven. On Maltsev digraphs. In *Proceedings of the 6th International Computer Science Symposium in Russia*, 2011. To appear.
6. Víctor Dalmau. Constraint satisfaction problems in non-deterministic logarithmic space. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP '02, pages 414–425. Springer-Verlag, 2002.
7. Víctor Dalmau and Andrei Krokhin. Majority constraints have bounded pathwidth duality. *Eur. J. Comb.*, 29(4):821–837, 2008.
8. Víctor Dalmau and Benoit Larose. Maltsev + Datalog → symmetric Datalog. In *LICS*, pages 297–306, 2008.
9. László Egri, Andrei Krokhin, Benoit Larose, and Pascal Tesson. The complexity of the list homomorphism problem for graphs. *Theory of Computing Systems (Special Issue on STACS 2010)*. To appear.
10. László Egri, Benoît Larose, and Pascal Tesson. Symmetric Datalog and constraint satisfaction problems in logspace. In *LICS*, pages 193–202, 2007.
11. Tomás Feder. Classification of homomorphisms to oriented cycles and of k-partite satisfiability. *SIAM J. Discrete Math.*, 14(4):471–480, 2001.
12. Tomas Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1999.
13. Erich Grädel. Capturing complexity classes by fragments of second-order logic. *Theor. Comput. Sci.*, 101(1):35–57, 1992.
14. D. Hobby and R.N. McKenzie. *The Structure of Finite Algebras*, volume 76 of *Contemporary Mathematics*. American Mathematical Society, Providence, R.I., 1988.
15. Neil Immerman. *Descriptive complexity*. Graduate Texts in Computer Science. Springer, 1999.
16. Benoit Larose and Pascal Tesson. Universal algebra and hardness results for constraint satisfaction problems. *Theor. Comput. Sci.*, 410(18):1629–1647, 2009.
17. Benoit Larose and László Zádori. Bounded width problems and algebras. *Algebra Universalis*, 56(3-4):439–466, 2007.
18. Leonid Libkin. *Elements of finite model theory*. Springer, 2004.
19. T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings of STOC*, pages 216–226, 1978.
20. Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.

Non-Definability Results for Randomised First-Order Logic

Kord Eickmeyer

Humboldt-Universität zu Berlin
Under den Linden 6
10099 Berlin, Germany

Abstract

We investigate the expressive power of randomised first-order logic (BPFO) on restricted classes of structures. While BPFO is stronger than FO in general, even on structures with a built-in addition relation, we show that BPFO is not stronger than FO on structures with a unary vocabulary, nor on the class of equivalence relations. The same techniques can be applied to show that evenness of a linear order, and therefore graph connectivity, can not be defined in BPFO. Finally, we show that there is an FO[≤]-definable query on word structures which can not be defined in BPFO[+1].

1998 ACM Subject Classification F.4.1 Model theory; F.1.2 Probabilistic computation

Keywords and phrases Descriptive complexity, randomised logics, derandomisation

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.218

1 Introduction

In [5], we introduced randomised logics as a tool for analysing randomised complexity classes using descriptive complexity theory. Randomised algorithms can be defined from deterministic ones by introducing a second input, namely a string of random bits whose length depends only on the length of the input and which is drawn uniformly at random from the set of all strings of that length. The outcome of such an algorithm $\mathbb{A}$ may then depend both on its input and on the particular choice of the random string, and for each fixed input x we get a certain acceptance probability, say $p_{\mathbb{A}}(x)$.

To define randomised complexity classes, one restricts attention to algorithms which have a probability gap, i.e., there is a certain interval $(\alpha, \beta] \subseteq [0, 1]$ such that $p_{\mathbb{A}}(x) \notin (\alpha, \beta]$ for all inputs x. Such an algorithm is said to accept its input if $p_{\mathbb{A}}(x) > \beta$. By parallel repetition and thresholding, this gap may be amplified, so that the definition of, say, randomised polynomial time or randomised logspace is very robust under the choice of the interval $(\alpha, \beta]$ (cf. [1]). However, if one does not demand any probability gap, the resulting complexity class PP becomes rather powerful, as witnessed by Toda's theorem [13] stating that P^{PP} contains the full polynomial hierarchy.

In [5], we defined randomised first-order logic BPFO in a similar manner by introducing additional relation symbols which are interpreted randomly. This way, we can define the satisfaction probability $\Pr(A \models \varphi)$ of a sentence φ in a structure A, and just like in the case of randomised algorithms we demand this to be outside of some interval $(\alpha, \beta]$ for all finite structures A. We then say that $A \models \varphi$ if this probability is $> \beta$ (see section 3 for details). Barrington et al.'s famous result that FO captures dlogtime-uniform AC^0 on structures with addition and multiplication easily carries over to the randomised world, i.e., one obtains a logic capturing dlogtime BPAC^0 on such structures. Similarly, randomised least fixed-point logic BPLFP captures BPP on ordered structures. Equipped with very

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 218–232

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

weak counting abilities, one also obtains a logic capturing BPP on all structures, albeit one with an undecidable syntax.

Previous research on the expressive power of logics on random structures mostly dealt with finite relational structures in which *all* relations were defined randomly. In some cases, the underlying universe was assumed to be ordered, but the ordering was not accessible to the logic. This holds true, for example, of the various 0-1-laws for first-order and infinitary logics [8, 6], which imply that derandomisation is possible on structures over the empty vocabulary. While these results have been generalised to probability distributions other than uniform (cf. [11]), hardly any work has been done on structures with random as well as non-random relations. Reasoning about partly random structures appears to require much more powerful tools, and the only previous work in this direction which we are aware of is by Shelah [10] and Boppana and Spencer [2], who prove what they call *smoothness laws* for ordered random structures, i.e., they only consider the case where the non-random part is a linear order. While there is no convergence law in this case, Boppana and Spencer prove that for every first-order sentence φ,

$$|\Pr(\mathcal{O}_n \models \varphi) - \Pr(\mathcal{O}_{n+1} \models \varphi)| = O\left(\frac{\log^d n}{n}\right),$$

where d is the quantifier depth of φ. We use essentially the same proof technique to show that BPFO can be derandomised on structures with a unary vocabulary and on equivalence classes; with the minor adjustment that we allow for arbitrary random relations instead of just random undirected graphs, their results imply theorem 8(a). Our application of that technique in proving Lemma 5 is complicated by the fact that we consider, for the non-random part, any structure defined over a unary vocabulary.

In contrast to randomised complexity classes such as BPP, for which there is evidence towards the fact that they can be derandomised (i.e., BPP = PTIME, cf. [9]), first-order logic provably gains expressive power by randomisation. In [5], we obtained the following results:

- on additive structures, $\mathsf{BPFO} \not\preceq \mathsf{FO}$, i.e., there is a query of additive structures which is definable in BPFO but not in FO
- on ordered structures, $\mathsf{BPFO} \not\preceq \mathsf{MSO}$
- $\mathsf{BPFO} \not\preceq \mathsf{C}^{\omega}_{\infty\omega}$ (infinitary counting logic)

On the other hand, we obtained the derandomisation results that $\mathsf{BPFO} \preceq \mathsf{MSO}$ on additive structures and $\mathsf{BPFO} \preceq \Sigma_2$ on all structures (both of which are basically translations of the Sipser-Gács-Lautemann-Theorem that $\mathsf{BPP} \subseteq \Sigma^p_2$) and $\mathsf{BPFO} \preceq \mathsf{FO}$ on structures over the empty vocabulary.

There is an elaborate machinery of tools for proving non-definability results in classical logics, most importantly game theoretic methods such as Ehrenfeucht-Fraïssé games and applications of Håstad's Switching lemma to first-order logic via a translation to AC^0 circuit families. To apply these methods to show that, say, the class of all connected graphs is not definable in first-order logic, one constructs, for each sentence φ, a pair of graphs G and G' such that $G \models \varphi \Leftrightarrow G' \models \varphi$, but only exactly one of the two is connected.

For proving non-definability in randomised logics, however, one has to prove that certain sentences can not have a probability gap. Therefore one has to investigate the behaviour of these sentences on *all* finite structures. For example, let A be a $\{P, R\}$-structure, where P is a unary relation and R a binary relation. We view the relation R as a function

$$f : \begin{cases} V(A) & \to 2^{P(A)} \\ a & \mapsto \{b \in P(A) \mid (a, b) \in R(A)\} \end{cases}$$

from the universe of A to subsets of $P(A)$ (cf. Figure 1). The following sentence in $\mathsf{FO}[\{P,R\}]$ is satisfied iff f is injective:

$$\varphi_{\text{inj}} = \forall x \forall y \exists z \, (Pz \wedge \neg(Rxz \leftrightarrow Ryz))$$

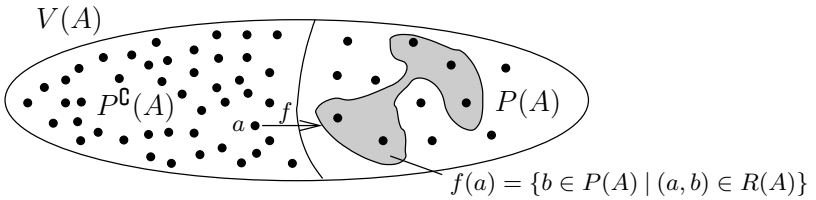


Figure 1 The random relation R interpreted as a function.

Up to isomorphism, a $\{P\}$-structure A is determined by its total number of element n and the number of elements k in $P(A)$. Now fix a $\{P\}$-structure A and let X be a randomly chosen $\{P,R\}$-expansion of A. The probability that f as defined above is injective is monotonely decreasing in n for fixed k and monotonely increasing in k for fixed n. In fact, because the range of the function f doubles if k is increased by 1, for almost all n this probability makes a sudden jump from nearly 0 for $k \leq k_n$ to nearly 1 at $k > k_n + 1$ for some k_n. In this sense, φ_{inj} *almost* has a gap. In [5] we used a similar sentence together with a binary relation to impose additional structures on $V(A)$ and $P(A)$ which can not be of size n and k such that $\Pr(A \models \varphi)$ is in $(0.2, 0.5)$.

In the present paper we show that binary relation symbols are actually necessary for this: On the class of all structures over vocabularies of only unary relations, BPFO is not more expressive than FO. For our above example this implies that for every $0 < \alpha < \beta < 1$, there is a $\{P\}$-structure A such that $\Pr(A \models \varphi_{\text{inj}}) \in (\alpha, \beta)$. Our proof uses a result of Boppana [3] on the average sensitivity of AC^0-circuits; a similar approach has been taken in [2] to proof smoothness laws for first-order logic.

In section 6, we then investigate the question of how expressive BPFO is on word models, i.e., structures in which all non-unary relations depend only on the size of the structure. Let Σ be a finite alphabet. With every word $w \in \Sigma^*$ we associate a structure which has one universe element for each position in w. The vocabulary of the structure contains one unary predicate P_a for each $a \in \Sigma$, along with some relations which only depend on the length of w. Two common choices for these relations are

- a binary successor relation, which we denote by $+1$ or $y \doteq x + 1$ and which is supposed to hold true iff y is the position immediately to the right of x, and
- a binary linear ordering relation $\leq$, where $x \leq y$ is supposed to hold true iff x is to the left of or identical to y.

The expressive power of various logics on these word models has been the subject of intensive study, cf. [12] for a comprehensive overview. As for complexity theory, while MSO-model-checking on word models is fixed-parameter tractable when parameterised by the size of the formula, this is not the case for general structures unless PTIME = NP.

When we speak of $\mathsf{FO}[+1]$, $\mathsf{FO}[\leq]$, $\mathsf{BPFO}[+1]$, and $\mathsf{BPFO}[\leq]$, we mean (randomised) first-order logic restricted to word models of the appropriate type. The very low expressive power of first-order logic on word models suggests that, as in the case of BPFO on unary structures, it might not be possible to ensure a probability gap on all finite structures (or at least on all word models) to get a BPFO-definable query on word models which is not definable in FO. As a first step in this direction, we show that there is an $\mathsf{FO}[\leq]$-definable query which can not be defined in $\mathsf{BPFO}[+1]$.

2 Preliminaries

We consider only finite structures over relational vocabularies. That is, a vocabulary σ is a finite set of relation symbols, each with an associated arity $r > 0$. A σ-structure A is a finite set $V(A)$ together with a subset $R(A) \subseteq V(A)^r$ for each relation symbol $R \in \sigma$ of arity r. An isomorphism $f : A \xrightarrow{\sim} B$ is a bijective function $f : V(A) \to V(B)$ such that for all r-ary $R \in \sigma$,

$$(a_1, \dots, a_r) \in R(A) \text{ iff } (f(a_1), \dots, f(a_r)) \in R(B),$$

and two structures A and B are called isomorphic (written $A \cong B$) if such an isomorphism exists. A *query* $\mathcal{Q}$ is a class of structures closed under isomorphisms. A partial isomorphism $a_1 \dots a_k \mapsto b_1 \dots b_k$ consists of k elements $a_1, \dots, a_k \in V(A)$ and k elements $b_1, \dots, b_k \in V(B)$ such that

$$(a_{i_1}, \dots, a_{i_r}) \in R(A) \text{ iff } (b_{i_1}, \dots, b_{i_r}) \in R(B)$$

for every r-ary $R \in \sigma$ and $1 \leq i_1, \dots, i_r \leq k$. For vocabularies $\sigma \subseteq \tau$, a *τ-expansion* of a σ-structure A is any τ-structure B for which $V(B) = V(A)$ and $R(B) = R(A)$ for all $R \in \sigma$.

First-order (FO) formulas are built from atomic formulas $x \doteq y$ and $Rx_1 \dots x_r$ for r-ary $R \in \sigma$ by boolean junctors, existential and universal quantification. The models relation $\models$, free variables, and quantifier depth of a formula are defined as usual. A sentence is a formula without free variables. For an FO-sentence φ, we denote by $\mathrm{Mod}(\varphi)$ the class of all finite structures A with $A \models \varphi$. A query $\mathcal{Q}$ is said to be definable in FO if there is a sentence φ such that $\mathcal{Q} = \mathrm{Mod}(\varphi)$.

Two structures A and B are called m-equivalent, written $A \equiv_m B$, if they satisfy exactly the same FO-formulas of quantifier rank up to m. By Ehrenfeucht's Theorem (cf. [4]), this is equivalent to the existence of a winning strategy for Duplicator in the following game (called Ehrenfeucht-Fraïssé game):

Two players, called Spoiler and Duplicator, take turns in choosing elements from two structures A and B. Spoiler moves first. If, in the k-th round, Spoiler chooses an element a_k from structure A, Duplicator has to answer with an element b_k from structure B, and vice versa. Duplicator wins if, after m rounds have been played, $a_1 \dots a_m \mapsto b_1 \dots b_m$ is a partial isomorphism.

After fixing a linear order on σ, a σ-structure A may be encoded (non-uniquely) by a string $x_A \in \{0,1\}^*$ of length polynomial in $|V(A)|$, by encoding the information $(a_1, \dots, a_r) \in R(A)$ for every tuple $(a_1, \dots, a_r) \in V(A)^r$ and every relation symbol $R \in \sigma$ by one letter. An FO-sentence φ of quantifier depth d may be translated into a family of boolean circuits $(C_n)_{n \geq 1}$ of depth d and size $n^{O(1)}$ such that

$$A \models \varphi \text{ iff } C_{|A|}(x_A) = 1,$$

and the outcome of $C_{|A|}(x_A)$ is independent of the particular string representing A. The circuits C_n are composed of negation gates and $\vee$ and $\wedge$ gates of arbitrary fan-in.

A result of Boppana gives a bound on the sensitivity of such circuit families:

▶ **Theorem 1.** *Let $f : \{0,1\}^* \to \{0,1\}$ be a boolean function computable by a family $(C_n)_{n \geq 0}$ of boolean circuits of depth d, size $n^{O(1)}$ consisting of negation gates and $\vee$ and $\wedge$-gates of unbounded fan-in. If x is chosen uniformly at random from $\{0,1\}^n$ then*

$$\mathbb{E}\left|\{1 \leq i \leq n \mid f(x) \neq f(x^{(i)})\}\right| \leq O(\log^{d-1} n),$$

where $x^{(i)}$ is the string x with the i-th bit flipped.

The expected value in the theorem is called the *average sensitivity* of f. For a proof, see [3].

3 Randomised logics

We briefly review the definition of randomised logics given in [5]. Throughout this section, let τ and ρ be disjoint vocabularies. Relations over ρ will be "random", and we will reserve the letter R for relation symbols from ρ. We are interested in *random* $(\tau \cup \rho)$*-expansions* of τ-structures. For a τ-structure A, by $\mathcal{X}(A, \rho)$ we denote the class of all $(\tau \cup \rho)$-expansions of A. We view $\mathcal{X}(A, \rho)$ as a probability space with the uniform distribution. Note that we can "construct" a random $X \in \mathcal{X}(A, \rho)$ by deciding independently for all k-ary $R \in \rho$ and all tuples $\vec{a} \in V(A)^k$ with probability $1/2$ whether $\vec{a} \in R(X)$. We are mainly interested in the probabilities

$$\Pr_{X \in \mathcal{X}(A,\rho)}(X \models \varphi)$$

that a random $(\tau \cup \rho)$-expansion of a τ-structure A satisfies a sentence φ of vocabulary $\tau \cup \rho$ of some logic. For brevity, we denote the above probability by $\Pr(A \models \varphi)$ whenever the vocabulary ρ of random relations is clear from the context.

▶ **Definition 2.** Let L be a logic and $0 \leq \alpha \leq \beta \leq 1$.

1. A formula $\varphi \in \mathsf{L}[\tau \cup \rho]$ that defines a k-ary query has an $(\alpha, \beta]$-*gap* if for all τ-structures A and all $\vec{a} \in V(A)^k$ it holds that

   $$\Pr(A \models \varphi[\vec{a}]) \leq \alpha \qquad \text{or} \qquad \Pr(A \models \varphi[\vec{a}]) > \beta.$$

2. The logic $\mathsf{P}_{(\alpha,\beta]}\mathsf{L}$ is defined as follows: For each vocabulary τ,

   $$\mathsf{P}_{(\alpha,\beta]}\mathsf{L}[\tau] := \bigcup_{\rho} \big\{ \varphi \in \mathsf{L}[\tau \cup \rho] \mid \varphi \text{ has an } (\alpha, \beta]\text{-gap} \big\},$$

   where the union ranges over all vocabularies ρ disjoint from τ. To define the semantics, let $\varphi \in \mathsf{P}_{(\alpha,\beta]}\mathsf{L}[\tau]$ be a sentence (the definition for arbitrary formulas is straightforward). Let ρ be such that $\varphi \in \mathsf{L}[\tau \cup \rho]$. Then for all τ-structures A,

   $$A \models \varphi \quad :\Leftrightarrow \quad \Pr(A \models \varphi) > \beta,$$

   and $\mathrm{Mod}(\varphi)$ is the class of all structures A with $A \models \varphi$.

It is easy to see that for every logic L and all α, β with $0 \leq \alpha \leq \beta \leq 1$ the logic $\mathsf{P}_{(\alpha,\beta]}\mathsf{L}$ is a well-defined logic, in the sense that the $\models$-relation is invariant under isomorphisms of the structure and under renamings and extensions of the vocabulary (see [5] for details). We will be focusing on the logic

$$\mathsf{BPFO} := \mathsf{P}_{(1/3,2/3]}\mathsf{FO}$$

in this paper. The strength of this logic does not depend on the exact choice of the parameters α and β, which justifies the arbitrary choice of the constants $1/3, 2/3$ in the definition. As for first-order logic, we say that a query $\mathcal{Q}$ is *definable* in BPFO if there is a sentence $\varphi \in \mathsf{BPFO}$ with $\mathcal{Q} = \mathrm{Mod}(\varphi)$.

4 BPFO = FO on structures with unary vocabulary

In [5] we gave several examples of queries which were definable in BPFO but (in particular) not in FO. A common feature of these queries is that they are defined on structures over a vocabulary with at least binary relations. In this section we will prove that this is in fact necessary:

▶ **Theorem 3.** *Let $\tau = \{P_1, \dots, P_s\}$ be a vocabulary containing only unary relations, and let $\varphi \in \mathsf{BPFO}$. Then there is a (non-randomised) $\mathsf{FO}[\tau]$-sentence defining the same query as φ.*

We may restrict ourselves to structures in which every element satisfies exactly one of the P_i, and we call these *τ-coloured* structures. In fact, a τ-structure can be seen as a set partitioned into 2^s classes, where the elements in each class satisfy exactly the same predicates P_i. We introduce a new vocabulary $\tau' = \{P'_I \mid I \subseteq [s]\}$ and associate with each τ-structure a τ'-coloured structure and vice versa in the obvious way. Similarly, each atomic formula $P_i x$ can be expressed as a boolean combination of atomic formulas $P'_I x$ and vice versa.

Up to isomorphism, a (finite) τ-coloured structure is described uniquely by a tuple $\vec{n} = (n_1, \dots, n_s) \in \mathbb{N}^s$ of non-negative integers giving the size of each class, and we will denote structures by such tuples. We denote the size of such a structure by $\|\vec{n}\| := \sum_{i=1}^{s} n_i$. For each $k \in \mathbb{N}$ we define an equivalence relation $\sim_k$ on $\mathbb{N}^s$ by saying $\vec{n} \sim_k \vec{m}$ iff

$$n_i = m_i \qquad \text{or} \qquad n_i \geq k \text{ and } m_i \geq k$$

for all $1 \leq i \leq s$. Then $\sim_k$ gives exactly the expressive power of first-order sentences of quantifier rank k on τ-coloured structures:

▶ **Lemma 4.** *Let φ be an $\mathsf{FO}[\tau]$-sentence of quantifier rank $\leq k$. Then on τ-coloured structures, $\mathrm{Mod}(\varphi)$ is a union of $\sim_k$-equivalence classes. Conversely, every union of $\sim_k$-equivalence classes can be defined by an $\mathsf{FO}[\tau]$-sentence of quantifier rank $\leq k$.*

Proof. This is a standard application of Ehrenfeucht-Fraïssé games, see, e.g., [4, ex. 2.3.12]. ◀

We may thus restate Theorem 3 as follows:

▶ **Lemma 5.** *Let $\tau = \{P_1, \dots, P_s\}$ be as above and let ρ be any relational vocabulary with $\tau \cap \rho = \emptyset$. Then for every $\varphi \in \mathsf{FO}[\tau \cup \rho]$ and $0 < \alpha < \beta < 1$ one of the following holds:*

1. *there is a tuple $(n_1, \dots, n_s) \in \mathbb{N}^s$ with*

   $$\Pr(A \models \varphi) \in (\alpha, \beta)$$

   or
2. *there is a $k \in \mathbb{N}$ such that for all $\vec{n}, \vec{m}$ with $\vec{n} \sim_k \vec{m}$ the probabilities $\Pr(\vec{n} \models \varphi)$ and $\Pr(\vec{m} \models \varphi)$ are either both $\leq \alpha$ or both $\geq \beta$.*

The proof of this lemma is based on the fact that, if we make a large colour class a little smaller by removing one element, the satisfaction probability of an $\mathsf{FO}[\tau \cup \rho]$-sentence does not change by much. Here, *large* means both absolutely large (at least a certain number of elements) and relatively large, i.e., containing at least some constant fraction of all elements. This is made precise in the following lemma, which we prove below:

▶ **Lemma 6.** *Let $\tau = \{P_1, \dots, P_s\}$ and ρ be vocabularies as above, and $\varphi \in \mathsf{FO}[\tau \cup \rho]$. For every $c, \epsilon > 0$ there is a $k = k_{c,\epsilon,\varphi} \in \mathbb{N}$ such that the following holds: If $\vec{n} \in \mathbb{N}^s$ is a tuple such that $n_i \geq c\,\|\vec{n}\|$ and $n_i \geq k$, then*

$$|\Pr(\vec{n} \models \varphi) - \Pr(\vec{n}' \models \varphi)| < \epsilon,$$

where $n'_i = n_i - 1$ and $n'_j = n_j$ for $j \neq i$.

Proof of Lemma 5. Let φ be any $\mathsf{FO}[\tau \cup \rho]$-sentence and let $k = k_{1/s,\beta-\alpha,\varphi}$ be the constant which Lemma 6 yields for $c = 1/s$ and $\epsilon = \beta - \alpha$. For any tuple $\vec{n} = (n_1, \ldots, n_s) \in \mathbb{N}^s$, the tuple $\vec{\nu}$ with

$$\nu_i = \min\{n_i, k\}$$

is a canonical representative of its $\sim_k$-equivalence class. We give a sequence

$$\vec{n} = \vec{n}_0, \vec{n}_1, \ldots, \vec{n}_l = \vec{\nu}$$

of tuples such that $\vec{n}_i \sim_k \vec{n}_{i+1}$ and

$$|\Pr(\vec{n}_i \models \varphi) - \Pr(\vec{n}_{i+1} \models \varphi)| < \beta - \alpha$$

hold for all $0 \leq i < l$. We define such a sequence by successively decreasing one of the maximal entries which are greater than k until there are no such entries left. Because any maximal entry of a tuple $\vec{n} \in \mathbb{N}^s$ must be at least $\|\vec{n}\|/s$, Lemma 6 precisely states that the satisfaction probability of φ never changes by more than $\beta - \alpha$ in each step, as claimed.

But now the satisfaction probabilities $\Pr(\vec{n}_i \models \varphi)$ along the sequence are either all $\leq \alpha$, all $\geq \beta$, or one of them is in the open interval (α, β). Because $\vec{\nu}$ is the same for all tuples in a $\sim_k$-equivalence class, the statement of the theorem follows. ◀

Notice that there may well be $\vec{n}$ and $\vec{m}$ with $\vec{n} \sim_k \vec{m}$ and such that $|\Pr(\vec{n} \models \varphi) - \Pr(\vec{m} \models \varphi)|$ is arbitrarily close to 1, but in that case, for every $\Pr(\vec{n} \models \varphi) < \alpha < \beta < \Pr(\vec{m} \models \varphi)$ we can find a $\vec{u}$ with $\Pr(\vec{u} \models \varphi) \in (\alpha, \beta)$.

Proof of lemma 6. We introduce a new unary relation symbol Q and define an $\mathsf{FO}[\tau \cup \rho \cup \{Q\}]$-formula ψ by restricting all quantifiers of φ to $Q \cup \bigcup_{j \neq i} P_j$. That is, we define ψ recursively from φ by

- if $\varphi = \exists x \varphi'$ then $\psi := \exists x(Qx \vee \bigvee_{j \neq i} P_j x) \wedge \psi'$,
- if $\varphi = \forall x \varphi'$ then $\psi := \forall x\big((Qx \vee \bigvee_{j \neq i} P_j x) \to \psi'\big)$,
- if $\varphi = \neg\varphi'$, then $\psi := \neg\psi'$,
- if $\varphi = \varphi' \vee \varphi''$, then $\psi := \psi' \vee \psi''$,
- if $\varphi = \varphi' \wedge \varphi''$, then $\psi := \psi' \wedge \psi''$, and
- $\psi := \varphi$ otherwise.

Define $\vec{m}$ by

$$m_i := 2n_i \qquad \text{and} \qquad m_j := n_j \text{ for } j \neq i.$$

Treating Q as a random relation (along with the relations in ρ) and conditioning on the size of $Q \cap P_i$ we get

$$\Pr(\vec{n} \models \varphi) = \Pr_{X \in \mathcal{X}(\vec{m}, \rho \cup \{Q\})} \big(X \models \psi \;\big|\; |Q \cap P_i| = n_i\big)$$

and

$$\Pr(\vec{n}' \models \varphi) = \Pr_{X \in \mathcal{X}(\vec{m}, \rho \cup \{Q\})} \big(X \models \psi \;\big|\; |Q \cap P_i| = n_i - 1\big).$$

Our goal is to show that these two (conditional) probabilities are not too far apart. We first translate the sentence ψ into a bounded-depth, polynomial-size circuit C as in Figure 2. The depth d of this circuit is equal to the quantifier depth of ψ, and it has one input for each relation symbol in $\rho \cup \{Q\}$ and each tuple of universe elements of appropriate arity. (We

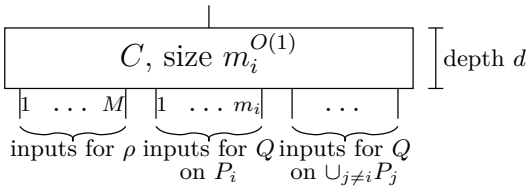


Figure 2 A polynomial-size, bounded-depth circuit for ψ

assume the unary predicates $P_1, \ldots, P_s$ to be hard-wired into the circuit.) In particular, there are $m_i = 2n_i$ inputs which determine the set $Q \cap P_i$.

The inputs corresponding to $Q \cap \bigcup_{j\neq i} P_j$ are, by our construction of ψ, irrelevant and we fix them to 0. Suppose there are M inputs corresponding to random relations in ρ. For each way of fixing these inputs to a certain value $y \in \{0,1\}^M$ we get a circuit C_y on m_i inputs, which is of the same depth as C. Furthermore, because $M = \|\vec{n}\|^{O(1)}$ and we assumed n_i to be $\Omega(\|\vec{n}\|)$, the size of C_y is polynomial in m_i.

By Theorem 1, the average sensitivity of C_y is polylogarithmic in n_i, and therefore also in m_i. This means that if $Q \subseteq [m_i]$ and $q \in [m_i]$ are chosen uniformly and independent of each other, then

$$\Pr(C_y(Q) \neq C_y(Q \triangle \{q\})) < \frac{(\log m_i)^{O(1)}}{m_i} < m_i^{-0.9}$$

for m_i large enough. Notice that Boppana's upper bound depends only on the size and depth of the C_y and thus it is independent of the particular choice of y.

Let A be the event that both $|Q \cap P_i| = n_i$ and $q \in Q$. Then

$$\Pr(A) = \frac{1}{2^{2n_i+1}} \binom{2n_i}{n_i},$$

which is $\Theta(n_i^{-1/2})$ and therefore $\Theta(m_i^{-1/2})$ by standard calculations (see, e.g., [7]). By the independence of the inputs of C we have

$$\Pr(\vec{n} \models \varphi) = 2^{-M} \sum_y \Pr\left(C_y(Q) = 1 \mid A\right)$$

and

$$\Pr(\vec{n}' \models \varphi) = 2^{-M} \sum_y \Pr\left(C_y(Q \triangle \{q\}) = 1 \mid A\right)$$

We may now bound the difference of these probabilities as follows:

$$\begin{aligned}
&|\Pr(\vec{n} \models \varphi) - \Pr(\vec{n}' \models \varphi)| \\
&\leq 2^{-M} \sum \Pr\left(C_y(Q) \neq C_y(Q \triangle \{q\}) \;\middle|\; A\right) \\
&\leq 2^{-M} \sum \frac{\Pr\left(C_y(Q) \neq C_y(Q \triangle \{q\}) \cap A\right)}{\Pr(A)} \\
&\leq 2^{-M} \sum \frac{\Pr\left(C_y(Q) \neq C_y(Q \triangle \{q\})\right)}{\Pr(A)} \\
&\leq m_i^{-0.9} \cdot \Theta(m^{1/2}) < m_i^{-0.3}
\end{aligned}$$

for m_i large enough. We assumed $m_i \geq k$, and thus this difference is $< \epsilon$ if we choose k large enough. ◀

The above proof technique can be adapted to yield the following somewhat stronger result:

▶ **Theorem 7.** *Let $\sigma = \{E\}$ be a vocabulary containing just one binary relation E, and let $\mathcal{EQ}$ be the class of all finite structures A for which $E(A)$ is an equivalence relation. Then* BPFO = FO *on $\mathcal{EQ}$.*

▶ Remark. Note that because $\mathcal{EQ}$ is definable in FO, for every sentence φ with a probability gap on $\mathcal{EQ}$ there is a sentence φ' which is equivalent to φ on $\mathcal{EQ}$ and has a probability gap on all finite structures.

Proof. Up to isomorphism, a structure $A \in \mathcal{EQ}$ is determined by a function $f^A : \mathbb{N} \to \mathbb{N}$ such that $f^A(s)$ counts the number of equivalence classes of size s (so that $|V(A)| = \sum s f^A(s) =: \|f\|$). For each $k \in \mathbb{N}$ we define a function

$$f_k^A(s) = \begin{cases} \min\{k, f^A(s)\} & \text{if } s < k, \\ \min\{k, \sum_{i \geq k} f^A(i)\} & \text{if } s = k, \\ 0 & \text{if } s > k. \end{cases}$$

We say $A \sim_k B$ if $f_k^A(s) = f_k^B(s)$ for all $s \in \mathbb{N}$. By standard techniques, a query $\mathcal{Q} \subseteq \mathcal{EQ}$ is definable in FO iff it is a union of $\sim_k$-equivalence classes for some k. A function f is *k-canonical* if $f(s) \leq k$ for all s and $f(s) = 0$ for all $s > k$. The k-canonical functions form a system of representatives for the equivalence relation $\sim_k$, and we denote the representative equivalent to f by $\tilde{f}$.

For notational convenience, again we assume there is only one random relation symbol R. Fix a formula $\varphi \in \{E, R\}$ and an $\epsilon > 0$. As in Lemma 6 we show that there is a k such that for every f there is a sequence

$$f = f_0 \sim_k f_1 \sim_k f_2 \sim_k \cdots \sim_k f_l = \tilde{f}$$

with $|\Pr(f_i \models \varphi) - \Pr(f_{i+1} \models \varphi)| < \epsilon$ along the sequence. To get from f_i to f_{i+1} we proceed as follows: Suppose $n := \|f_i\| > k^3$. If one equivalence class has $> n^{1/3}$ elements (i.e. $f_i(s) > 0$ for some $s > n^{1/3}$) we remove one element from that class. Otherwise, there must be an $s \leq n^{1/3}$ such that $f(s) > n^{1/3}$.In this case, remove an entire equivalence class of size s. Finally, if $\|f_i\| \leq k^3$, we may remove elements from equivalence classes of size $> k$ and remove an equivalence class of size s if there are more than k classes of that size. Proceeding in this way we eventually reach $\tilde{f}$.

Removing an element from a class is done by randomly choosing from a class of twice the size, and removing a class of a certain size is done by randomly choosing among twice as many classes of that size. We defer details to the full version of this paper. ◀

5 Some queries which are not definable in BPFO

Using the same techniques as in the proof of Theorem 3, we obtain the following non-definability results:

▶ **Theorem 8.** *The following queries on finite structures are not definable in* BPFO*:*

(a) Over the vocabulary $\{\leq\}$ containing a binary relation symbol $\leq$, the query "$\leq$ defines a linear order of even cardinality"

(b) Over the vocabulary $\{E\}$ containing a binary relation symbol E, the query "E defines a connected graph"

(c) Over the vocabulary $\{+1\}$ containing a binary relation symbol $+1$, the query "the universe elements form an initial segment of the natural numbers, treating $+1$ as a successor relation".

Proof. Denote by $\mathcal{O}_n$ the linear order on n elements. For query (a), introduce a new random unary relation P on a linear order of length $2n$ and relativise all quantifiers to P as in the proof of Lemma 6. Letting n tend to infinity, this shows that

$$\left| \Pr_{\mathcal{O}_n,\rho}(X \models \varphi) - \Pr_{\mathcal{O}_{n-1},\rho}(X \models \varphi) \right| \to 0$$

for any $\mathsf{FO}[\{\leq\} \cup \rho]$-sentence φ. In a different context, this result had already been obtained by Boppana and Spencer [2], using essentially the same argument.

Non-definability of query (b) follows because we can define a graph on $\mathcal{O}_n$ in FO which is connected iff n is even. Namely, identifying the elements of the linear order with the first n natural numbers, connect elements

- x and $x+2$ for all $1 \leq x \leq n-2$,
- 2 and $n-1$.

Thus a BPFO-sentence defining connected graphs could be used to define evenness of a linear order (see [4] for details). A similar argument works for query (c). ◀

6 Randomised First-Order Logic on Words

As before, we denote by $\mathsf{FO}[+1]$, $\mathsf{FO}[\leq]$, $\mathsf{BPFO}[+1]$, and $\mathsf{BPFO}[\leq]$ (randomised) first-order logic restricted to word models of the appropriate type. There are two natural definitions of BPFO on restricted classes of structures, namely one which demands BPFO sentences to have a gap on *all* finite structures, and one which demands this only on structures from the restricted class. Because the fact that $\leq$ defines a linear order is definable in FO, word models of the second type can be defined in FO and this distinction does not affect the expressive power of $\mathsf{BPFO}[\leq]$. In contrast to this, the successor relation $+1$ can not be defined in FO, because connexness of the transitive closure of $+1$ is not definable. By Theorem 8(c), this holds true also for BPFO. Therefore, the two definitions of $\mathsf{BPFO}[+1]$ potentially have different expressive power. Our counterexample in Theorem 9 works for both variants.

The expressive power of $\mathsf{FO}[+1]$ and $\mathsf{FO}[\leq]$ is well understood, see [12]. In particular, the query

$$Q := a^*ba^*ca^* \subseteq \{a,b,c\}^*$$

of all words which contain exactly one b to the left of exactly one c and an arbitrary number of as is not definable in $\mathsf{FO}[+1]$. It is easily seen to be definable in $\mathsf{FO}[\leq]$ by the sentence

$$\exists x \exists y (P_b x \wedge P_c y \wedge x \leq y \wedge \forall z (P_a z \vee z \doteq x \vee z \doteq y)).$$

We show that Q is not definable in $\mathsf{BPFO}[+1]$:

▶ **Theorem 9.** *There is no $\mathsf{BPFO}[+1]$-sentence φ such that*

$$w \models \varphi \quad \Leftrightarrow \quad w \in Q$$

for all $w \in \{a,b,c\}^*$.

Proof. Let $\sigma = \{+1, P_a, P_b, P_c\}$ be the vocabulary of our word models. We show the theorem by exhibiting a sequence of pairs of words v_n, w_n such that

(i) $v_n \in Q$, $w_n \notin Q$ for all $n \geq 1$ and
(ii) for every vocabulary ρ disjoint from σ and every $\mathsf{FO}[\sigma \cup \rho]$-sentence φ,

$$|\Pr(v_n \models \varphi) - \Pr(w_n \models \varphi)| \to 0 \qquad (n \to \infty).$$

In fact, choosing

$$v_n = a^n b a^n c a^n \qquad w_n = a^n c a^n b a^n$$

will do. Condition (i) is obviously satisfied. For condition (ii), let ρ be disjoint from σ and let φ be a sentence of quantifier rank r. The successor relation induces a distance measure on the elements of the structures, which we denote by d; we assume $d(x, y) = 1$ if $x = y + 1$ or $y = x + 1$. We denote by d_r the bounded distance function

$$d_r(x, y) := \begin{cases} d(x, y) & \text{if } d(x, y) \leq r \\ \infty & \text{otherwise.} \end{cases}$$

By $S^r(x)$ we denote the r-ball around an element x in (a $(\sigma \cup \rho)$-expansion of) a word structure A, i.e.,

$$S^r(x) := \{y \in V(A) \mid d(x, y) \leq r\},$$

and if $a_1, \ldots, a_k$ are elements of $V(A)$, then $A|_{S^r(a_1,\ldots,a_k)}$ denotes the induced substructure of A on the union $\bigcup_{i=1}^{k} S^r(a_k)$ of the r balls around these elements. We say that two sets $U, V \subseteq V(A)$ *touch* if there are $x \in U$ and $y \in V$ with $x = y + 1$ or $y = x + 1$.

For $n > 3^r$, the word structures v_n and w_n satisfy exactly the same first-order sentences of quantifier rank up to r. A winning strategy for the r-move Ehrenfeucht-Fraïssé-game on v_n and w_n can be given explicitly as follows: For ease of notation, we denote the first and the last position of v_n by a_1 and a_2, the unique position containing a b by a_3 and that containing a c by a_4, and likewise for $b_1, \ldots, b_4$. Suppose after k moves, elements $a_5, \ldots, a_{k+4}$ have been chosen in v_n, and elements $b_5, \ldots, b_{k+4}$ have been chosen in w_n. Assume Spoiler chooses an element a in v_n. Throughout the game, Duplicator maintains the property that

$$d_{3^{r-k}}(a_i, a_j) = d_{3^{r-k}}(b_i, b_j) \tag{1}$$

for $1 \leq i, j \leq k + 4$. Notice that this property holds before the first move (i.e., for $a_1, \ldots, a_4$ and $b_1, \ldots, b_4$) if $n > 3^r$. Let $r' = r - k - 1$ be the number of rounds remaining after the k-th move.

(I) If a is in $v_n|_{S^{3^{r'}}(a_1,\ldots,a_{k+4})}$, then choose the corresponding element in w_n, i.e., the unique element $b \in V(w_n)$ which has

$$d_{3^{r'}}(a_i, a) = d_{3^{r'}}(b_i, b)$$

for $1 \leq i \leq k + 4$. This is possible because if $d(b_i, b), d(b_j, b) \leq 3^{r'}$, then $d(b_i, b_j) \leq 2 \cdot 3^{r'} < 3^{r-k}$ and $d_{3^{r-k}}(a_i, a_j) = d_{3^{r-k}}(b_i, b_j)$ by property (1).
(II) Otherwise, choose any element of w_n which has distance $> 3^{r'}$ from all elements $b_1, \ldots, b_{k+4}$.

Duplicator's answer if Spoiler chooses an element b in w_n is determined analogously. After r rounds have been played, the map $a_i \mapsto b_i$ is a partial isomorphism, because all relations in σ are determined by d_1-distances. This is because on the words v_n and w_n, the relations P_a, P_b and P_c depend only on the d_1-distance from u and v, which are parts of the tuples.

We now extend this strategy to random expansions X of v_n and Y of w_n. Let

$$c_0 := 1, \qquad c_{i+1} := 4r_i + 2.$$

In the game on X and Y, Duplicator maintains the stronger property that after the k-th move,

$$X_k := X|_{S^{c_r-k}(a_1,\dots,a_{k+4})} \cong Y|_{S^{c_r-k}(b_1,\dots,b_{k+4})} =: Y_k, \tag{2}$$

treating the a_is and b_is as constants. That this, there is an isomorphism $f : X_k \xrightarrow{\sim} Y_k$ such that $f(a_i) = b_i$ for $1 \leq 1 \leq k+4$. This is of course not possible for all random expansions: At the very least, the random expansions have to agree on the c_r-balls around min, max, u and v. If this is the case, then with very high probability Duplicator can indeed maintain property (2), as we will now show. The argument resembles the proof of the classical 0-1-law for first-order logic (cf. [4]), but it involves some more housekeeping to deal with the additional structure introduced by the +1-relation.

Let μ_w denote the uniform probability measure on the set $\mathcal{X}(w,\rho)$, i.e.,

$$\mu_w(V) := \frac{|V|}{|\mathcal{X}(w,\rho)|}$$

for $V \subseteq \mathcal{X}(w,\rho)$. For ease of notation we drop the subscript w. Let s be the number of non-isomorphic $(\sigma \cup \rho)$-expansions of $v_{2c_r+2}|_{S^{c_r}(\min,\max,u,v)}$, and let $A_1,\dots,A_s$ be structures representing these isomorphism types. Notice that the four c_r-balls which make up the universe of this substructure do not touch, as is the case in all v_n and w_n for large enough n. We let $V_n^{(j)}$ be the set of all $(\sigma \cup \rho)$-expansions X of v_n with

$$X|_{S^{c_r}(\min,\max,u,v)} \cong A_j,$$

and analogously for $W_n^{(j)}$. If the c_r-balls around min, max, u and v do not touch, then the induced substructures of v_n and w_n on the union of these balls are isomorphic. Thus for large enough n, the $V_n^{(j)}$ ($W_n^{(j)}$) form a partition of $\mathcal{X}(v_n,\rho)$ ($\mathcal{X}(w_n,\rho)$), and

$$\mu\left(V_n^{(j)}\right) = \mu\left(W_n^{(j)}\right) = \frac{1}{s}.$$

For any two structures $X \in V_n^{(j)}$ and $Y \in W_n^{(j)}$, the tuples $a_1,\dots,a_4$ and $b_1,\dots,b_4$ as defined above satisfy property (2). We now show that there are subsets $\hat{V}_n^{(j)} \subset V_n^{(j)}$ and $\hat{W}_n^{(j)} \subset W_n^{(j)}$ such that Duplicator can maintain property (2) for r moves on structures taken from these subsets.

To be precise, we define Duplicator's strategy if Spoiler chooses a from structure X as follows:

(I) If a is in $X|_{S^{2c_{r'}+1}(a_1,\dots,a_{k+4})}$, then choose the corresponding element in Y, i.e., the unique element $b \in V(Y)$ which has

$$d_{c_{r'}}(a_i, a) = d_{c_{r'}}(b_i, b)$$

for $1 \leq i \leq k+4$. These are exactly the a whose $c_{r'}$-ball touches the $c_{r'}$-ball around some previously chosen a_i.

(II) Otherwise, choose any element of Y which has distance $> 2c_{r'} + 1$ from all elements $b_1, \dots, b_{k+4}$. Thus the $c_{r'}$-ball around the newly chosen element touches no $c_{r'}$-ball around a previously chosen element.

Moves of type (I) in the above strategy can always be carried out by Duplicator and maintain property (2). Moves of type (II) can only fail if there is a tuple $b_1, \dots, b_{k+4}$ in Y and a $(\sigma \cup \rho)$-structure Z containing elements $a_1, \dots, a_{k+4}$ and a such that

- $Z \in \mathcal{X}(v_n, \rho)$,
- $Z|_{S^{c_{r-k}}(a_1,\dots,a_{k+4})} \cong Y|_{S^{c_{r-k}}(b_1,\dots,b_{k+4})}$,
- $d(a, a_i) > 2c_{r'} + 1$ for $1 \leq i \leq k+4$, and
- $Z|_{S^{c_{r'}}(a_1,\dots,a_{k+4},a)} \ncong Y|_{S^{c_{r'}}(b_1,\dots,b_{k+4},b)}$ for all $b \in V(Y)$.

Let $m := 3n + 2 = |V(Y)|$. There are $O(m^r)$ many possible tuples $b_1, \dots, b_{k+4}$, and for each such tuple, there are only constantly (depending only on ρ) many choices for Z and $a_1, \dots, a_{k+4}, a$ with non-isomorphic $Z|_{S^{c_{r'}}(a_1,\dots,a_{k+4},a)}$. But for each of these $O(m^r)$ possibilities, there is a subset $M \subset V(Y)$ with

- $|M| = \Omega(n)$,
- $d(b, b_i) > 2c_{r'} + 1$, for each $b \in M$ and $1 \leq i \leq k+4$, and
- $d(b, b') > 2c_{r'} + 1$ for every $b, b' \in M$.

Because the $c_{r'}$-balls around the elements of M do not overlap, each of the elements in M satisfies

$$Z|_{S^{c_{r'}}(a_1,\dots,a_{k+4},a)} \cong Y|_{S^{c_{r'}}(b_1,\dots,b_{k+4},b)}$$

independently with some probability $p > 0$ depending only on r' and ρ. The probability that none of the $b \in M$ satisfies this is therefore $(1-p)^{|M|} = e^{-\Omega(n)}$, and by a union bound, there is a subset $\hat{W}_n^{(j)} \subset WV_n^{(j)}$ with

$$\mu\big(\hat{W}_n^{(j)}\big) = \big(1 - o(1)\big)\mu\big(W_n^{(j)}\big)$$

and such that on structures $Y \in \hat{W}_n^{(j)}$, Duplicator can maintain property (2) for r many moves when challenged to move in Y. A subset $\hat{V}_n^{(j)} \subset V_n^{(j)}$ can be defined analogously.

But now we have defined disjoint sets $\hat{V}_n^{(1)}, \dots, \hat{V}_n^{(s)} \subset \mathcal{X}(v_n, \rho)$ and $\hat{W}_n^{(1)}, \dots, \hat{W}_n^{(s)} \subset \mathcal{X}(w_n, \rho)$ such that

(a) $\left|\mu(\hat{V}_n^{(j)}) - \mu(\hat{W}_n^{(j)})\right| \to 0$ for $n \to \infty$ and all $1 \leq j \leq s$,

(b) $\mu\left(\bigcup_j \hat{V}_n^{(j)}\right) \to 1$ for $n \to \infty$

(c) for every n and j, if $X \in \hat{V}_n^{(j)}$ and $Y \in \hat{W}_n^{(j)}$, then $X \cong_r Y$.

This implies that for every $\mathsf{FO}[\sigma \cup \rho]$-sentence φ,

$$|\Pr(v_n \models \varphi) - \Pr(w_n \models \varphi)| \to 0$$

as $n \to \infty$, and therefore Q is not definable in $\mathsf{BPFO}[+1]$. ◀

7 Conclusion

We have shown non-definability results for randomised first-order by bounding the difference

$$|\Pr(A \models \varphi) - \Pr(B \models \varphi)|$$

for certain pairs of σ-structures A and B and $\mathsf{FO}[\sigma \cup \rho]$-sentences φ. We did so using two very different tools:

- Boppana's result on the average sensitivity of bounded-depth polynomial size circuits, and
- Ehrenfeucht-Fraïssé-games on (partially) random structures.

These two approaches have very different strengths and weaknesses: The Ehrenfeucht-Fraïssé-game approach worked well on the query Q because all but a finite number of positions in each of the strings v_n and w_n looked exactly the same to any FO-sentence of quantifier rank $\leq r$. This is not the case in the two-coloured structure with colour-class sizes n and $\log n$, for example. This approach might be extended by drawing the random expansions of A and B from a well-chosen joint distribution.

In order to apply Boppana's result to bound the difference

$$|\Pr(A \models \varphi) - \Pr(B \models \varphi)|$$

between the acceptance probability of φ in two structures A and B, we defined a larger structure within which we were able to define a structure C (using an additional random relation) such that $C \cong A$ with probability at least $n^{-1+\epsilon}$, and such that changing the additional random relation on one tuple resulted in $C \cong B$ with high probability. With this method we could bound the above difference for enough pairs of structures to actually derandomise BPFO on structures with a unary vocabulary completely. This approach was made possible by the fact that the structures A and B for which we applied it had lots of automorphisms, making it easy to define them within the bigger structure with high probability.

It seems reasonable to conjecture that $\mathsf{BPFO}[+1]$ can be derandomised to $\mathsf{FO}[+1]$. This is because to an FO-sentence of quantifier rank r, two positions in the string which are further apart than 3^r are completely non-related, and thus it should be possible to generate chains of strings $w_0, \ldots, w_l$ by only changing small parts in each step to get a version of Lemma 5 for strings. However, neither the Ehrenfeucht-Fraïssé-game approach nor the approach using Boppana's lemma seem to suffice for this.

Acknowledgements

The author would like to thank Martin Grohe and Nicole Schweikardt for helpful discussions on this research topic, and an anonymous referee for pointing out the similarities to the work of Boppana and Spencer [2]. Thanks also to Anuj Dawar for suggesting the extension of Theorem 3 to equivalence classes.

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity.* Cambridge University Press, 2009.

2 Ravi Boppana and Joel Spencer. Smoothness laws for random ordered graphs. In Ravi Boppana and James Lynch, editors, *Logic and Random Structures*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 15–32. American Mathematical Society, 1995.

3 Ravi B. Boppana. The average sensitivity of bounded-depth circuits. *Information Processing Letters*, 63(5):257–261, 1997.

4 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Perspectives in Mathematical Logic. Springer-Verlag, 2nd edition, 1999.

5 Kord Eickmeyer and Martin Grohe. Randomisation and derandomisation in descriptive complexity theory. In *Computer Science Logic*, volume 6247 of *LNCS*, pages 275–289. Springer-Verlag, 2010.

6 R. Fagin. Probabilities on finite models. *Journal of Symbolic Logic*, 41:50–58, 1976.

7 W. Feller. *An Introduction to Probability Theory and Its Aplications*, volume I. John Wiley & Sons, 1957.

8 Y.V. Glebskiĭ, D.I. Kogan, M.I. Liogon'kiĭ, and V.A. Talanov. Range and degree of realizability of formulas in the restricted predicate calculus. *Kibernetika*, 2:17–28, 1969. Englisch translation, Cybernetics 5:142–154,1969.

9 Russell Impagliazzo and Avi Wigderson. PTIME = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *STOC*, pages 220–229, 1997.

10 Saharon Shelah. Very weak zero one law for random graphs with order and random binary functions. *Random Structures & Algorithms*, 9(4):351–358, 1996.

11 Joel Spencer. *The Strange Logic of Random Graphs*. Springer, 2001.

12 Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994.

13 S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

System T and the Product of Selection Functions*

Martín Escardó[1], Paulo Oliva[2], and Thomas Powell[2]

1 **University of Birmingham**
Department of Computer Science
m.escardo@cs.bham.ac.uk
2 **Queen Mary University of London**
School of Electronic Engineering and Computer Science
{paulo.oliva, tpowell}@eecs.qmul.ac.uk

Abstract

We show that the finite product of selection functions (for all finite types) is primitive recursively equivalent to Gödel's higher-type recursor (for all finite types). The correspondence is shown to hold for similar restricted fragments of both systems: The recursor for type level $n+1$ is primitive recursively equivalent to the finite product of selection functions of type level n. Whereas the recursor directly interprets induction, we show that other classical arithmetical principles such as bounded collection and finite choice are more naturally interpreted via the product of selection functions.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Primitive recursion, product of selection functions, finite choice, dialectica interpretation

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.233

1 Introduction

In his 1958 paper published in the journal *dialectica* [7] Gödel introduced a novel interpretation of intuitionistic arithmetic into a quantifier-free calculus of functionals, the so-called system **T**. System **T** is essentially primitive recursive arithmetic PRA with the schema of recursion extended to all finite types $\{\mathbb{N}, \mathbb{N}\to\mathbb{N}, (\mathbb{N}\to\mathbb{N})\to\mathbb{N}, \dots\}$. Gödel's aim was to show that quantifier dependencies in arithmetic could be captured by this class of primitive recursive functionals, and therefore that the consistency of arithmetic could be reduced to that of system **T**.

In Gödel's dialectica interpretation, the recursors play a fundamental role in the interpretation of the induction axioms. Parsons [15] studied the precise relationship between the complexity of the type of the recursor and the logical complexity of the induction formula, establishing a correspondence between the well-known fragments of arithmetic based on restricted induction and fragments of system **T** based on restricted recursion.

The dialectica interpretation of arithmetic was quickly extended to classical analysis by Spector [17], via a new form of recursion on well-founded trees known as *bar recursion*. Spector's dialectica interpretation of arithmetical comprehension goes via the classical axiom of countable choice, which in turn is reduced to the double negation shift (see [17] for details)

$$\forall i \neg\neg A(i) \to \neg\neg \forall i A(i).$$

* This work was partially supported by Royal Society (2nd author) grant 516002.K501/RH/kk, and an EPSRC Doctoral Training Account (3rd author).

Hence, the computational interpretation of full classical analysis was reduced to the interpretation of this seemingly harmless (obviously true) principle.

The first two authors have recently shown [3, 2, 5] that Spector's bar recursion is primitive recursively equivalent to an unbounded iteration of the product of *selection functions*, a highly intuitive construction that has appeared over and over again in various guises in game theory, fixed point theory, algorithms, and proof theory. For further details see Section 3, their original paper [3] or a recent survey [6].

In [3] it is observed that just as an unbounded iteration of the product of selection functions provides an intuitive interpretation of the double negation shift, a finite iteration of the product directly interprets the *finite* double negation shift

$$\forall m(\forall i \leq m \neg\neg A(i) \to \neg\neg \forall i \leq m A(i)),$$

which in turn is closely related to a number of well-known 'set theoretic' principles such as finite choice and bounded collection. In Section 4 we show that just as the computational analogue of induction is Gödel's primitive recursion on all finite type, a natural computational analogue of *finite choice* is given by the product of selection functions. Furthermore, analogous to Parsons result for induction we establish a correspondence between the logical complexity of the choice formula and the complexity of the product.

It is well-known, however, that both bounded collection and finite choice are equivalent to number induction. More specifically, Parsons proved that the hierarchy of bounded collection axioms is strictly interleaving on the hierarchy of induction axioms [14]. Therefore it is natural to ask precisely how the product of selection functions is related to primitive recursion. Our main result (Section 5) is that Gödel's primitive recursor of type level $n+1$ is equivalent, over a weak base theory, to the finite product of selection functions of type level n. In particular, the finite product of selection functions of lowest types already defines the Ackermann function.

There are several advantages of considering this equivalent form of the Gödel primitive recursor R

1. Given the equivalence of the unbounded product of selection functions with Spector's bar recursion [3], the equivalence of the finite product with the recursor R provides for a smooth passage from the functional interpretation of arithmetic (finite fixed number of iterations) to that of analysis (finite but unbounded number of iterations). Therefore, we obtain the correspondence

$$\frac{\text{Finite product of selection functions}}{\text{Arithmetic}} = \frac{\text{Unbounded product of selection functions}}{\text{Analysis}}$$

   In this sense, the product of selection functions allows for a uniform transition from arithmetic to analysis. This is discussed further in Section 6.
2. The product of selection functions has a natural reading in terms of calculations of optimal strategies in sequential games. Hence, witnessing terms involving the product of selection functions (rather than R or Spector's bar recursion) can normally be given a clear intuitive meaning. The connection between games and the product is explained in Section 3.
3. Whereas the recursor directly interprets induction, we show that when interpreting other classical arithmetical principles closely related to bounded collection and finite choice it is the finite product of selection functions which allows for a direct, and hence more illuminating, interpretation. See Section 4 for details.

2 Fragments of Arithmetic

We take as a base theory the standard induction-free fragment of arithmetic Q (cf. [1]), which contains axioms for the non-logical symbols 0, S, $+$, $\cdot$ and $\leq$. It is well known that we can construct a hierarchy of strong fragments of arithmetic by adding induction axioms, or alternatively choice or collection axioms, to our base theory. The axiom *scheme of induction* is specified as

$$\text{IND}: \ A_0 \wedge \forall i{<}m\,(A_i \to A_{i+1}) \to A_m.$$

The *scheme of finite choice* is specified as

$$\text{FAC}: \ \forall i{\leq}m\,\exists x A_i(x) \to \exists s \forall i{\leq}m\, A_i(s_i).$$

The *scheme of bounded collection* is specified as

$$\text{BC}: \ \forall i{\leq}m\,\exists x A_i(x) \to \exists k \forall i{\leq}m\,\exists x{\leq}k\, A_i(x)$$

As usual, if S is one of our schemata, Σ_n-S (Π_n-S) denotes S restricted to Σ_n (Π_n) formulas.

▶ **Definition 1** (Fragments of arithmetic)**.** In this paper we will consider the following fragments of classical arithmetic.

1. The weak theory PA_0 consists of the base theory Q plus induction restricted to quantifier-free formulas.
2. The theories $\text{I}\Sigma_n$, $\text{F}\Sigma_n$, $\text{B}\Sigma_n$ consist of PA_0 plus the Σ_n-IND, Σ_n-FAC, Σ_n-BC schema respectively. The theories $\text{I}\Pi_n$, $\text{F}\Pi_n$ and $\text{B}\Pi_n$ are defined similarly.
3. Full Peano arithmetic PA consists of PA_0 plus induction for all formulas (or, equivalently as we will discuss, finite choice or bounded collection for all formulas).

It is easy to show (cf. [13, 14]) that PA_0 proves the equivalences

$$\begin{aligned} \Sigma_n\text{-IND} &\leftrightarrow \Pi_n\text{-IND}, \\ \Sigma_{n+1}\text{-FAC} &\leftrightarrow \Pi_n\text{-FAC}, \text{ and} \\ \Sigma_{n+1}\text{-BC} &\leftrightarrow \Pi_n\text{-BC} \end{aligned}$$

for all n. Hence, the fragments $\text{I}\Sigma_n$ and $\text{I}\Pi_n$ are equivalent, similarly $\text{F}\Sigma_{n+1} = \text{F}\Pi_n$ and $\text{B}\Sigma_{n+1} = \text{B}\Pi_n$.

It has also been shown by Sieg [16] that $\text{F}\Pi_n$ and $\text{B}\Pi_n$ are equivalent. Although the fragment based on bounded collection $\text{B}\Pi_n$ is more widely used, we will see that the dialectica interpretation of finite choice is slightly more natural and direct than that of bounded collection.

The precise relationship between the fragments of arithmetic based on induction and those based on choice and collection principles was established by Parsons [14] and Paris and Kirby [13].

▶ **Theorem 2** ([13, 14])**.** *Let $T \subseteq S$ mean that every theorem of T is a theorem of S. Then*

1. $\text{B}\Pi_n \subseteq \text{I}\Sigma_{n+1}$ *but* $\text{I}\Sigma_{n+1} \nsubseteq \text{B}\Pi_n$
2. $\text{I}\Sigma_n \subseteq \text{B}\Pi_n$ *but* $\text{B}\Pi_n \nsubseteq \text{I}\Sigma_n$.

Proof. These results are collected together in [13]. We remark that $\text{I}\Sigma_{n+1} \nsubseteq \text{B}\Pi_n$ was discovered independently by Lessan [12]. ◀

2.1 Fragments of Gödel's system T

In this section we recall some basic facts about Gödel's dialectica interpretation. As we mentioned in the introduction, Gödel showed that intuitionistic arithmetic can be interpreted in the quantifier-free system **T**. Combining the dialectica interpretation with the usual negative translation allows us to reduce full classical arithmetic to **T**. For that we shall normally take Kuroda's negative translation $A \mapsto A^N \equiv \neg\neg A^*$, which places double negations after universal quantifiers and in front of the whole formula [11].

We are interested in fragments of system **T** that correspond, under the dialectica interpretation, to the fragments of arithmetic discussed above. We take as a base theory the fragment $\mathbf{T}_0$ in which recursion is restricted to type 0.

▶ **Definition 3** (Fragment $\mathbf{T}_0$)**.** We work in a many-sorted language in which the set of types is defined inductively, containing the type $\mathbb{N}$ of natural numbers, function types $X \to Y$ (also written as Y^X), product types $X \times Y$ and types X^* representing finite sequences of elements of type X. As usual, the degree of each type is defined inductively as

$$\begin{aligned} \deg(\mathbb{N}) &:= 0 \\ \deg(X \to Y) &:= \max\{\deg(X)+1, \deg(Y)\} \\ \deg(X \times Y) &:= \max\{\deg(X), \deg(Y)\} \\ \deg(X^*) &:= \deg(X). \end{aligned}$$

The set of terms of $\mathbf{T}_0$ are those of the simply typed λ-calculus with finite products and function types, plus constants for all functions definable using primitive recursion of type 0. The axioms of $\mathbf{T}_0$ consist of:

1. standard axioms of classical propositional logic, axioms for (fully extensional) equality, substitution and induction,
2. defining axioms for each constant symbol.

Notation. We will denote the operation of concatenating a finite sequence $s\colon X^*$ with an infinite sequence $\alpha\colon X^{\mathbb{N}}$ as $s * \alpha\colon X^{\mathbb{N}}$. We will use the same notation also when concatenating two finite sequences, or appending an element to a sequence. For $q\colon X^{\mathbb{N}} \to R$ and $s\colon X^*$ we write $q_s\colon X^{\mathbb{N}} \to R$ for the function $q_s(\alpha) = q(s * \alpha)$.

We obtain extensions of $\mathbf{T}_0$ by adding constant symbols for higher type recursion together with their defining axioms. For any type X the recursor R^X of type $X \to (\mathbb{N} \to X \to X) \to \mathbb{N} \to X$ has defining axioms:

$$\begin{aligned} \mathsf{R}^X_0(y,z) &\stackrel{X}{=} y \\ \mathsf{R}^X_{n+1}(y,z) &\stackrel{X}{=} z(n, \mathsf{R}^X_n(y,z)) \end{aligned}$$

where $y\colon X$ and $z\colon \mathbb{N} \to X \to X$.

▶ **Definition 4** (Fragments of system **T**)**.** We consider the following well-known extensions of $\mathbf{T}_0$.

1. The theory $\mathbf{T}_n$ consists of $\mathbf{T}_0$ plus recursors R^X and their defining axioms for all types X with $\deg(X) \leq n$.
2. System **T** consists of $\mathbf{T}_0$ plus recursors of all finite types.

Gödel's dialectica interpretation interprets each formula A of Heyting arithmetic as a formula A^D of the form $\exists x \forall y A_D(x,y)$, where A_D is a quantifier-free formula in the language of $\mathbf{T}$ and x, y are tuples of potentially higher type. For details of the translation, the reader is referred to [1].

▶ **Theorem 5** ([7]). *Gödel's key results are the following:*

1. *If Heyting arithmetic proves the formula A, then there is a sequence of terms t in system $\mathbf{T}$ such that $\mathbf{T}$ proves $A_D(t,y)$.*
2. *If Peano arithmetic proves the formula A, then there is a sequence of terms t such that $\mathbf{T}$ proves $(A^N)_D(t,y)$, where A^N denotes the negative translation of A.*

The recursors in $\mathbf{T}$ are essentially only required to interpret the *non-logical* axioms of arithmetic, and form a natural functional analogue of induction. There is in fact a precise correspondence under the dialectica interpretation between the arithmetic hierarchy $\mathrm{I}\Sigma_n$ and the functional hierarchy $\mathbf{T}_n$, which is a consequence of the following lemma.

▶ **Lemma 6.** *Given an arbitrary formula A, suppose that $(A^N)^D = \exists x \forall y (A^N)_D(x,y)$. Then for $n > 1$:*

1. *if A is a Π^0_n formula then the tuple x contains variables of degree at most $n-1$ and the tuple y contains variables of degree at most $n-2$;*
2. *if A is a Σ^0_n formula then the tuple x contains variables of degree at most n and the tuple y contains variables of degree at most $n-1$.*

Proof. Simple induction on n. ◀

▶ **Theorem 7** ([15]). *The functional interpretation of Π_n-IND only requires primitive recursion of level $n-1$. Therefore the fragment of arithmetic III_n (or equivalently $\mathrm{I}\Sigma_n$) is interpreted in $\mathbf{T}_{n-1}$, in the sense that if III_n proves A then there is a sequence of terms t in $\mathbf{T}_{n-1}$ such that $\mathbf{T}_{n-1}$ proves $(A^N)_D(t,y)$.*

Proof. Follows from Lemma 6. See [15] for details. ◀

3 The Product of Selection Functions

In [5], a *selection function* is defined to be any function of type $(X \to R) \to X$. The intuition behind the name is that we view functions $X \to R$ as predicates over the type X (where R is interpreted as a set of truth values), and the selection function as a choice procedure that for each predicate selects some element of X. Selection functions are closely related to the notion of *generalised quantifiers*, functionals of type $(X \to R) \to R$, in the sense that every selection function ε is associated with a quantifier $\bar{\varepsilon} p := p(\varepsilon p)$.

Notation. We abbreviate the types $(X \to R) \to X$ by $J_R X$, and the types $(X \to R) \to R$ by $K_R X$.

▶ **Example 8.** 1. By the law of excluded middle, for any non-empty type X we have $\forall p \exists y^X (\exists x^X p(x) \Rightarrow p(y))$. Hence, by the axiom of choice there exists a selection function $\varepsilon \colon J_{\mathbb{B}} X$ that satisfies $\exists x\, p(x) \Leftrightarrow p(\varepsilon p)$ for any logical predicate $p \colon X \to \mathbb{B}$ (this is similar to Hilbert's ε operator of the ε-calculus). Similarly, there is also a selection function δ such that $\forall x\, p(x) \Leftrightarrow p(\delta p)$ for all p. These selection functions are associated, respectively, with the usual logical quantifiers $\exists, \forall \colon K_{\mathbb{B}} X$.

2. By the extreme value theorem there exists a selection function argsup: $J_{\mathbb{R}}[0,1]$ that for any continuous function $f\colon [0,1] \to \mathbb{R}$ returns a point at which f attains it supremum i.e. $\sup(f) = f(\text{argsup} f)$. The selection function arginf: $J_{\mathbb{R}}[0,1]$ is defined similarly. These selection functions are associated with the quantifiers $\sup, \inf\colon K_{\mathbb{R}}[0,1]$.

The theory of generalised quantifiers and selection functions is explored in detail in [3, 5, 6], where in particular, a product operation on selection functions is defined. The main achievement of these papers has been to demonstrate that this product of selection functions is an extremely versatile construction that appears naturally in several different areas of mathematics and computer science, such as fixed point theory (Bekič's lemma), game theory (backward induction), algorithms (backtracking), and proof theory (bar recursion).

For the rest of this section we shall define this product of selection functions, and explain how it has an intuitive meaning in terms of optimal plays in sequential games. In Section 4 we then show how the number theoretic principle of finite choice is naturally (dialectica) interpreted by this product.

▶ **Definition 9** (Binary product of selection functions, [5])**.** Given selection functions $\delta\colon J_R X$ and $\Delta\colon J_R X^{\mathbb{N}}$ and a functional $q\colon X^{\mathbb{N}} \to R$, let

$$\begin{aligned} A(x^X) &\overset{X^{\mathbb{N}}}{:=} \Delta(\lambda\alpha.q_x(\alpha)), \\ a &\overset{X}{:=} \delta(\lambda x.q_x(A(x))), \end{aligned}$$

where $q_x(\alpha)$ abbreviates $q(x*\alpha)$. Then we define the binary product of the selection functions δ and Δ, denoted $\delta \otimes \Delta\colon J_R X^{\mathbb{N}}$, by

$$(\delta \otimes \Delta)(q) := a * A(a).$$

As described in [5], one can iterate the binary product above on a given sequence of selection functions. In this paper we will only consider the finite iteration of the binary product.

▶ **Definition 10** (Finite product of selection functions)**.** We define the *finite product of selection functions* for types (X, R), denoted $\mathsf{P}_i^{X,R}$, by the recursion schema

$$\mathsf{P}_i^{X,R}(\varepsilon)(m) \overset{J_R X^{\mathbb{N}}}{=} \begin{cases} \mathbf{0}^{J_R X^{\mathbb{N}}} & \text{if } i > m \\ \varepsilon_i \otimes \mathsf{P}_{i+1}^{X,R}(\varepsilon)(m) & \text{if } i \leq m \end{cases}$$

where $m \in \mathbb{N}$, $\mathbf{0}$ is the constant 0 functional of appropriate type, and ε_i are selection functions of type $J_R X$. Expanding the definition of the binary product (Definition 9) this is equivalent to the schema

$$\mathsf{P}_i^{X,R}(\varepsilon)(m)(q) \overset{X^{\mathbb{N}}}{=} \begin{cases} \mathbf{0}^{X^{\mathbb{N}}} & \text{if } i > m \\ a * \mathsf{P}_{i+1}^{X,R}(\varepsilon)(m)(q_a) & \text{if } i \leq m \end{cases}$$

where $a := \varepsilon_i(\lambda x.q_x(\mathsf{P}_{i+1}^{X,R}(\varepsilon)(m)(q_x)))$.

As opposed to in [5] here the finite product is taken over an infinite stream of selection functions. In what follows, where only $\varepsilon_0, \ldots, \varepsilon_m$ are specified it is implicit that the finite product $\mathsf{P}_i^{X,R}(\varepsilon)(m)$ is taken over a canonical extension of this finite sequence.

As an alternative to adding the recursors R to $\mathbf{T}_0$, as in Definition 4, we shall also consider extending $\mathbf{T}_0$ with the finite product operator P instead.

▶ **Definition 11.** 1. The theory $\mathbf{P}_n$ consists of $\mathbf{T}_0$ plus a symbol for the finite product of selection functions $\mathsf{P}^{X,R}$ and its defining axiom for all types X with $\deg(X) \leq n$.
2. $\mathbf{T}_0 + \mathsf{P}$ consists of $\mathbf{T}_0$ plus the finite product $\mathsf{P}^{X,R}$ for all finite types.

▶ Remark. The complexity of the type R has no effect on the recursive strength of $\mathsf{P}^{X,R}$, as one can show that $\mathsf{P}^{X,R}$, for arbitrary type R, is definable over $\mathbf{T}_0$ from $\mathsf{P}^{X,X^{\mathbb{N}}}$. Formally, given $\varepsilon_i \colon J_R X$ and $q \colon X^{\mathbb{N}} \to R$ define a new selection function $\varepsilon^q \colon J_{X^{\mathbb{N}}} X$ as

$$\varepsilon_i^q(P^{X \to X^{\mathbb{N}}}) \stackrel{X}{=} \varepsilon(\lambda x^X.q(P(x))).$$

We have that $\mathsf{P}(\varepsilon)(m)(q) = \mathsf{P}(\varepsilon^q)(m)(\mathrm{id})$, where $\mathrm{id} \colon X^{\mathbb{N}} \to X^{\mathbb{N}}$ is the identity functional.

The main result of this article is that $\mathbf{T}_0 + \mathsf{P}$ is equivalent to $\mathbf{T}_0 + \mathsf{R}$ (and hence to Gödel's system $\mathbf{T}$), and that more specifically there is a direct correspondence between the restricted fragments of both systems. But first we explain how selection functions and their finite product are fundamental in the study of sequential games.

3.1 Finite sequential games

One of the most interesting aspects of the product of selection functions is that it computes *optimal strategies* for a general class of sequential games. This concrete setting offers the most insight into how the product works, so we explain it briefly here.

▶ **Definition 12** (Finite sequential games)**.** An m-round sequential game is defined by a tuple (R, X, ε, q) where R and X are arbitrary types.

- X is the set of possible moves for any round. A play is a sequence $\alpha \colon X^m$.
- R is the set of possible outcomes.
- $q \colon X^m \to R$ is the outcome function that maps a play to its outcome.
- $\varepsilon_i \colon J_R X$ is the selection function for round $i \leq m$.

These kind of games have been defined in full generality in [3, 6], where in particular the set of moves may vary from one round to the other. Moreover, the games defined there allows for arbitrary *quantifiers* to describe the goal of each round. When these quantifiers have associated selection functions an optimal strategy for the game can be computed. Here, for simplicity, we will assume the selection functions are explicitly given in the definition of the game.

Also in [3, 6], the notions of optimal strategy and optimal play are defined. The intuition is as follows. We think of the selection functions ε_i as specifying at round i what the optimal move at that round would be if we knew the final outcome corresponding to each of the candidate moves, i.e. $p \colon X \to R$. The selection function takes this mapping of moves to outcomes and tells us what the "best" move would be in that particular case $\varepsilon_i(p) \colon X$. Now, a play is considered optimal if at all rounds the best move has indeed been played. That is to say that, there are functions p_i which compute the real outcome from the move being played, i.e. $p_i(\alpha(i)) = q\alpha$, and that $\alpha(i)$ is exactly what the selection function at round i would choose, i.e. $\alpha(i) = \varepsilon_i(p_i)$.

The main theorem of [6] is that the product of the given selection functions for each round, when applied to the outcome function, computes an optimal play α in the game.

▶ **Theorem 13** ([6])**.** *Given a sequential game as above, let*

$$\alpha := \mathsf{P}_0(\varepsilon)(m)(q).$$

Then, α is an optimal play in the sense that setting

$$p_i := \lambda x.q_{[\alpha](i)*x}\left(\mathsf{P}_{i+1}(\varepsilon)(m)(q_{[\alpha](i)*x})\right),$$

where $[\alpha](i)$ is the finite initial segment of α of length i, we have

$$\begin{aligned} \alpha(i) &\stackrel{X}{=} \varepsilon_i(p_i) \\ p_i(\alpha(i)) &\stackrel{R}{=} q\alpha \end{aligned} \tag{1}$$

for all $i \leq m$.

In more general terms, the equations (1) characterise the product as an operation that generates a state of equilibrium from a finite sequence of selection functions. An optimal strategy is one instance of such an equilibrium. The significance of the product of selection functions lies in the fact that these governing equations appear repeatedly in a variety of different contexts.

4 Interpreting the Principle of Finite Choice

In this section we show how the finite product of selection functions allows for a more direct interpretation of (the classical) finite choice and bounded collection principles. As observed by Spector [17], the interpretation of the negative translation of choice follows intuitionistically from choice itself given the double negation shift

$$\forall i \neg\neg A(i) \to \neg\neg \forall i A(i).$$

The same applies to finite choice, where the negative translation of finite choice follows from finite choice plus the finite double negation shift

$$\forall m(\forall i \leq m \neg\neg A(i) \to \neg\neg \forall i \leq m A(i)).$$

Contrary to the double negation shift, the *finite* double negation shift is provable in Heyting arithmetic, by induction on m. The proof, however, is rather intricate, and when interpreted (via the dialectica) leads to witnesses based on the recursor R which are difficult to grasp computationally. This is in stark contrast with the proof of the following theorem:

▶ **Theorem 14.** *The finite product of selection functions interprets (via the dialectica interpretation) the finite double negation shift.*

Proof. Assume $A(i)$ has dialectica interpretation $\exists x \forall y A_i(x, y)$. The (partial) dialectica interpretation of the finite double negation shift is equivalent to

$$\forall m(\exists \varepsilon \forall p \forall i \leq m A_i(\varepsilon_i p, p(\varepsilon_i p)) \to \forall q \exists \alpha \forall i \leq m A_i(\alpha(i), q\alpha)).$$

Given m, ε and q then taking α, p_i as in Theorem 13 we clearly have

$$A_i(\varepsilon_i p_i, p_i(\varepsilon_i p_i)) \to A_i(\alpha(i), q\alpha)$$

for all $i \leq m$, so the α, p_i computed directly via the product of selection function witness the interpretation of the double negation shift. ◀

It should be observed that when using modified realizability (instead of the dialectica interpretation) it is the so-called J-shift which is directly interpreted by the product of selection functions (cf. [4]). It is, therefore, rather interesting that when applying the dialectica interpretation the same product of selection functions allows for a direct interpretation of the double negation shift instead.

A closer look at the dialectica interpretation of the double negation shift sheds some light on why an operation that computes optimal plays in sequential games crops up in proof theory in this manner. The selection functions ε_i above act as realisers for the premise of the double negation shift, and as such can be seen as a collection of strategies $(\varepsilon_i)_{i<m}$ that for each i refute any counterexample functions p (in the sense of Kreisel [9, 10]) attempting to disprove the predicate A_i.

The functional interpretation of the double negation shift calls for a procedure that takes this collection of 'point-wise' strategies and produces a co-operative strategy in which the ε_i work together to refute a *global* counterexample function q attempting to disprove the predicate $\forall i < m\, A_i$. Such a procedure is provided naturally by the product of selection functions.

▶ **Corollary 15.** *The finite product of selection functions interprets the principle of finite choice.*

Proof. The negative translation of finite choice, assuming that A_i is Π^0_n, is equivalent to

$$\forall i \leq m \neg\neg\exists x A_i^*(x) \to \neg\neg\exists \alpha \forall i \leq m A_i^*(\alpha_i),$$

where A^* is obtained by placing double negations after each universal quantifier in A. This follows directly from the double negation shift applied to the formula $\exists x A_i^*(x)$, and its dialectica interpretation is precisely that of this double negation shift. Hence, the product of selection functions realises the dialectica interpretation of (the negative translation of) finite choice. ◀

Analogous to Theorem 7, we can extend Corollary 15 to fragments of arithmetic based on choice.

▶ **Theorem 16.** *The fragment of arithmetic* $\mathrm{F}\Pi_n$ *is interpreted in* $\mathbf{P}_{n-1}$.

Proof. This is clear for $n = 1$. For $n > 1$ by Lemma 6 if A_i is a Π^0_n formula the functional interpretation of $\exists x A_i^*(x)$ is of the form $\exists x, \tilde{x} \forall y (A_i^*)_D(x, \tilde{x}, y)$ where the variables of the tuple $\langle x^0, \tilde{x}\rangle$ have degree at most $n-1$ (note that $A^D \leftrightarrow A^*$ for Π^0_n formulas). Hence, by inspecting the proof of Theorem 14 we see that an instance of Π_n-FAC is interpreted in $\mathbf{P}_{n-1}$. ◀

Finally we remark that bounded collection and its consequences, such as the infinite pigeonhole principle (cf. [8], p. 173), are naturally interpreted by the product of selection functions in a similar manner, given that finite choice straightforwardly implies bounded collection. The realiser for the negative translation of bounded collection, namely

$$\forall i \leq m \neg\neg\exists x A_i^*(x) \to \neg\neg\exists k \forall i \leq m \neg\neg\exists x \leq k\, A_i^*(x)$$

based on the product of selection functions is obtained from that for finite choice by essentially applying the maximum operator to the first m elements of the sequence α (see [6] for more details).

5 The Recursor and the Product of Selection Functions

Although the recursor R directly interprets the induction schema, we have seen in Section 4 that it is the finite product of selection functions which directly interprets finite choice and bounded collection. As discussed in Section 2, Parson showed that the hierarchy of bounded collection axioms is strictly interleaving on the hierarchy of induction axioms. Therefore, one might conjecture that the hierarchy of finite products of selection functions would be also strictly interleaved in the hierarchy of Gödel's primitive recursors. In this section we show that this is not the case, and in fact the recursor of type level $n+1$ is primitive recursively equivalent to the finite product of selection functions of type level n.

▶ **Definition 17.** It will be convenient to make use of the functional $\mathsf{B}(\varepsilon)(m)(q)\colon X^* \to X^{\mathbb{N}}$ defined as

$$\mathsf{B}(\varepsilon)(m)(q)(s) := \mathsf{P}_{|s|}(\varepsilon)(m)(q_s).$$

By using the expanded definition of P (cf. Definition 10), it is easy to see that B, for fixed ε, m and q, satisfies the recursion schema

$$\mathsf{B}(s) := \begin{cases} \lambda n.\mathbf{0}^{\mathbf{X}} & \text{if } |s| > m \\ a_s * \mathsf{B}(s * a_s) & \text{if } |s| \leq m \end{cases}$$

where $a_s = \varepsilon_{|s|}(\lambda x.q(s * x * \mathsf{B}(s * x)))$. The intuitive reading of B is an operation that takes a partial play s and returns $s * \alpha$ where α is a continuation of s that is optimal up to round m. In particular, an easy induction argument proves that for all $i \leq m$

$$\mathsf{P}_0(\varepsilon)(m)(q)_i \equiv \mathsf{B}(\langle\,\rangle)_i = \mathsf{B}(x_0, \ldots, x_{i-1})_0 \tag{2}$$

where $x_j := \mathsf{P}_0(\varepsilon)(m)(q)_j$ for $j < i$. In what follows the parameters of B will always be clear from the context, so we will omit them for simplicity.

Notation. Given two fragments of $\mathbf{T}$, say $\mathbf{T}'$ and $\mathbf{T}''$, we write $\mathbf{T}' \Rightarrow \mathbf{T}''$ if all functionals definable in $\mathbf{T}''$ can be already defined in $\mathbf{T}'$.

▶ **Theorem 18.** *$\mathsf{P}^{X,R}$ is definable in $\mathbf{T}_0 + \mathsf{R}^{X^* \to X^{\mathbb{N}}}$, so in particular $\mathbf{T}_{n+1} \Rightarrow \mathbf{P}_n$.*

Proof. Looking at the definition of the finite product, namely

$$\mathsf{P}_i^{X,R}(\varepsilon)(m)(q) \overset{X^{\mathbb{N}}}{=} \begin{cases} \lambda n.\mathbf{0}^X & \text{if } i > m \\ a * \mathsf{P}_{i+1}^{X,R}(\varepsilon)(m)(q_a) & \text{if } i \leq m \end{cases}$$

where $a := \varepsilon_i(\lambda x.q_x(\mathsf{P}_{i+1}^{X,R}(\varepsilon)(m)(q_x)))$, it is clear that the schema is just a standard recursion of type $X^* \to X^{\mathbb{N}}$ in which the quantity $m+1-i$ decreases along the recursion until it reaches 0. Formally, define the functionals $y^{\varepsilon,q,m}\colon X^* \to X^{\mathbb{N}}$ and $z^{\varepsilon,q,m}\colon \mathbb{N} \times (X^* \to X^{\mathbb{N}}) \to (X^* \to X^{\mathbb{N}})$ parametrised by ε, q and m as

$$\begin{aligned} y^{\varepsilon,q,m}(s) &:= \lambda n.\mathbf{0}^X \\ z^{\varepsilon,q,m}(i, F^{X^* \to X^{\mathbb{N}}})(s) &:= a_s * F(s * a_s) \end{aligned}$$

where $a_s := \varepsilon_{m \dot{-} i}(\lambda x.q_{s*x}(F(s * x)))$ and $m \dot{-} i$ denotes truncated subtraction. We claim that for all s

$$\mathsf{B}(s) = \mathsf{R}_{m+1 \dot{-} |s|}^{X^* \to X^{\mathbb{N}}}(y^{\varepsilon,q,m}, z^{\varepsilon,q,m})(s),$$

where B is as in Definition 17. In particular, it would follow that

$$\mathsf{P}_0^{X,R}(\varepsilon)(m)(q) = \mathsf{B}(\langle\rangle) = \mathsf{R}_m^{X^* \to X^{\mathbb{N}}}(y^{\varepsilon,q,m}, z^{\varepsilon,q,m})(\langle\rangle).$$

The claim is proved by induction on $m + 1 \dot{-} |s|$. For $|s| \geq m + 1$ we have

$$\mathsf{R}_{m+1\dot{-}|s|}(s) = \mathsf{R}_0(s) = y^{\varepsilon,q,m}(s) = \lambda n.\mathbf{0}^X = \mathsf{B}(s).$$

Now for $|s| = i < m + 1$ we have

$$\mathsf{R}_{m+1\dot{-}|s|}(s) = \mathsf{R}_{(m\dot{-}i)+1}(s) = z^{\varepsilon,q,m}(m\dot{-}i, \mathsf{R}_{m\dot{-}i})(s) = a_s * \mathsf{R}_{m\dot{-}i}(s * a_s)$$

where $a_s := \varepsilon_i(\lambda x.q_{s*x}(\mathsf{R}_{m\dot{-}i}(s * x)))$. But by induction hypothesis we have that

$$\mathsf{R}_{m\dot{-}i}(s * x) = \mathsf{B}(s * x)$$

for all $|s| = i$. Therefore $\mathsf{R}_{m+1\dot{-}i}(s) = \mathsf{B}(s)$. This completes the induction. ◀

We will show that the the types above are optimal, in the sense that we also have the converse $\mathbf{P}_n \Rightarrow \mathbf{T}_{n+1}$. But before showing that, lets us first show how one easily has $\mathbf{P}_n \Rightarrow \mathbf{T}_n$. The stronger result will require a more involved argument.

▶ **Theorem 19.** R^X *is definable in* $\mathbf{T}_0 + \mathsf{P}^{X,X^{\mathbb{N}}}$*, so in particular* $\mathbf{P}_n \Rightarrow \mathbf{T}_n$*.*

Proof. Given arbitrary functionals $y\colon X$ and $z\colon \mathbb{N} \to (X \to X)$, define selection functions $\varepsilon^{y,z}\colon \mathbb{N} \to J_{X^{\mathbb{N}}}X$ parametrised by y and z as

$$\varepsilon_i^{y,z}(p^{X \to X^{\mathbb{N}}}) \overset{X}{:=} \begin{cases} y & \text{if } i = 0 \\ z(i-1, p(\mathbf{0}^X)_{i-1}) & \text{if } i > 0. \end{cases}$$

Clearly $\lambda y, z.\varepsilon^{y,z}$ can be constructed in $\mathbf{T}_0$. We prove by induction on m that

$$\mathsf{R}_m(y, z) = (\mathsf{B}(\varepsilon^{y,z})(m)(\mathrm{id})(\langle\rangle))_m,$$

where $\mathrm{id}\colon X^{\mathbb{N}} \to X^{\mathbb{N}}$ is the identity λ-term. We shall actually think of m as fixed and show

$$\mathsf{R}_i(y, z) = (\mathsf{B}(\varepsilon^{y,z})(m)(\mathrm{id})(\langle\rangle))_i,$$

for $i \leq m$, by induction on i. When $i = 0$ we have (abbreviating $\mathsf{B}(s) \equiv \mathsf{B}(\varepsilon^{y,z})(m)(\mathrm{id})(s)$)

$$\mathsf{B}(\langle\rangle)_0 = \varepsilon_0^{y,z}(\lambda x \ldots) = y = \mathsf{R}_0(y, z).$$

Assuming that $x_j = \mathsf{R}_j(y, z) = \mathsf{B}(\langle\rangle)_j$, for $j < i$. We have

$$\begin{aligned} \mathsf{B}(\langle\rangle)_i &\overset{(2)}{=} \mathsf{B}(x_0, \ldots, x_{i-1})_0 \\ &= \varepsilon_i(\lambda x.\langle x_0, \ldots, x_{i-1}, x\rangle * \mathsf{B}(x_0 \ldots, x_{i-1}, x)) = z(i-1, x_{i-1}) = \mathsf{R}_i(y, z). \end{aligned}$$

◀

Already, we obtain the following key result.

▶ **Corollary 20.** *Gödel's system* $\mathbf{T}$*, i.e.* $\mathbf{T}_0 + \mathsf{R}$*, can be equivalently defined as* $\mathbf{T}_0 + \mathsf{P}$*.*

The intuition behind the proof of Theorem 19 is that the type $R = X^{\mathbb{N}}$ represents a register of elements of type X on which we perform a computation. The selection function ε_i sets the entry x_i at position i on the register to be $z(i-1, x_{i-1})$, where x_{i-1} is the entry at position $i-1$. The product $\mathsf{P}_0(m)(\varepsilon_i)(\mathrm{id})$ carries out the first m steps of this computation, returning $\mathsf{R}_m(y,z)$ at position m.

Of course the finite product of selection functions is able to perform a variety of computations on a register $X^{\mathbb{N}}$ in this manner - in which the ε_i determine the entry in the ith position of the register. However, in general the selection functions are capable of making this decision based not only on the previous entries but depending on the effect that potential choices have on *subsequent* entries. This suggests that the finite product of type X is a more powerful construction than the recursor of type X.

▶ **Theorem 21.** *$\mathsf{R}^{X\to X}$ is definable in $\mathbf{T}_0 + \mathsf{P}^{X,X^{\mathbb{N}}}$.*

Proof. Given arbitrary functionals $y\colon X^X$ and $z\colon \mathbb{N} \to (X^X \to X^X)$, define selection functions $\varepsilon_i^{y,z,n,a}\colon J_{X^{\mathbb{N}}}X$ parametrised by $y\colon X^X$ and $z\colon \mathbb{N} \to (X^X \to X^X)$ and $n\colon \mathbb{N}$ and $a\colon X$ as

$$\varepsilon_i^{y,z,n,a}(p^{X\to X^{\mathbb{N}}}) :\overset{X}{=} \begin{cases} z(n-1, \lambda x.p(x)_1)(a) & \text{if } i = 0 \\ z(n-i-1, \lambda x.p(x)_{i+1})(p(\mathbf{0}^X)_{i-1}) & \text{if } 0 < i < n \\ y(p(\mathbf{0}^X)_{i-1}) & \text{if } i = n \\ \mathbf{0}^X & \text{otherwise} \end{cases}$$

for $n > 0$, and

$$\varepsilon_i^{y,z,0,a}(p^{X\to X^{\mathbb{N}}}) :\overset{X}{=} \begin{cases} y(p(a)_0) & \text{if } i = 0 \\ \mathbf{0}^X & \text{otherwise.} \end{cases}$$

The functional $\lambda y, z, n, a.\varepsilon^{y,z,n,a}$ can be constructed in $\mathbf{T}_0$ since we only make use of combinatory completeness and definition by cases (which is primitive recursive of level 0 and allowed in $\mathbf{T}_0$). We prove that R can be defined as

$$\mathsf{R}_n(y,z) = \lambda a.(\mathsf{P}_0(\varepsilon^{y,z,n,a})(n)(\mathrm{id}))_0.$$

This is trivial for $n = 0$, so in the following we assume that $n > 0$. Once again, for convenience we set $\mathsf{B}(s) := \mathsf{P}_{|s|}(\varepsilon^{y,z,n,a})(n)(\mathrm{id}_s)$. First, we claim that

$$(\mathsf{B}(x_0, \ldots, x_{i-1}))_0 = \mathsf{R}_{n-i}(y,z)(x_{i-1})$$

for all $0 < i \leq n$. We proceed by induction on $n - i$. For $i = n$ we have

$$(\mathsf{B}(x_0, \ldots, x_{n-1}))_0 = \varepsilon_n(\lambda x.\langle x_0, \ldots, x_{n-1}, x\rangle * \lambda n.\mathbf{0}^X) = y(x_{n-1}) = \mathsf{R}_0(y,z)(x_{n-1}).$$

For $0 < i < n$,

$$\begin{aligned} (\mathsf{B}(x_0, \ldots, x_{i-1}))_0 &= \varepsilon_i(\lambda x.\langle x_0, \ldots, x_{i-1}, x\rangle * \mathsf{B}(x_0, \ldots, x_{i-1}, x)) \\ &= z(n-i-1, \lambda x.\mathsf{B}(x_0, \ldots, x_{i-1}, x)_0)(x_{i-1}) \\ &= z(n-i-1, \lambda x.\mathsf{R}_{n-i-1}(y,z)(x))(x_{i-1}) \\ &= \mathsf{R}_{n-i}(y,z)(x_{i-1}), \end{aligned}$$

assuming, by hypothesis, that $\mathsf{B}(x_0,\dots,x_{i-1},x) = \mathsf{R}_{n-(i+1)}(y,z)(x)$. This proves the claim, and the theorem follows directly:

$$\begin{aligned}\mathsf{P}_0(\varepsilon)(n)(\mathrm{id})_0 &= \mathsf{B}(\langle\,\rangle)_0\\ &= \varepsilon_0(\lambda x.x * \mathsf{B}(x))\\ &= z(n-1,\lambda x.\mathsf{B}(x)_0)(a)\\ &= z(n-1,\lambda x.\mathsf{R}_{n-1}(y,z)(x))(a) = \mathsf{R}_n(y,z)(a).\end{aligned}$$

◀

▶ **Corollary 22.** $\mathbf{P}_n \Leftrightarrow \mathbf{T}_{n+1}$.

Proof. One direction is given by Theorem 18. The other follows from Theorem 21: It can be shown that any type of level n is isomorphic to the pure type of that level, and consequently any two recursors of the same type level are inter-definable over $\mathbf{T}_0$. If $\deg(X) = n$ then $\deg(X \to X) = n+1$, therefore by Theorem 21, $\mathbf{P}_n \Rightarrow \mathbf{T}_{n+1}$. ◀

Theorem 22 tells us that, in particular, the product of selection functions over the type X is strictly stronger than primitive recursion of type X. For the case $X = \mathbb{N}$ we can illustrate this directly by constructing the Ackermann function in $\mathbf{P}_0$.

▶ **Example 23** (Ackermann function)**.** Define the selection functions $\varepsilon_i^{n,a}\colon J_{\mathbb{N}^{\mathbb{N}}}\mathbb{N}$ parametrised by natural numbers n and a as

$$\varepsilon_i^{n,a}(p^{\mathbb{N}\to\mathbb{N}^{\mathbb{N}}}) :\overset{\mathbb{N}}{=} \begin{cases} (\lambda x.p(x)_1)^{(a+1)}(1) & \text{if } i=0\\ (\lambda x.p(x)_{i+1})^{(p(\mathbf{0}^X)_{i-1}+1)}(1) & \text{if } 0<i<n\\ p(\mathbf{0}^X)_{i-1}+1 & \text{if } i=n\\ 0 & \text{otherwise}\end{cases}$$

for $n > 0$, and

$$\varepsilon_i^{0,a}(p^{\mathbb{N}\to\mathbb{N}^{\mathbb{N}}}) :\overset{\mathbb{N}}{=} \begin{cases} p(a)_0+1 & \text{if } i=0\\ 0 & \text{otherwise}\end{cases}$$

where $f^{(j)}$ is defined in $\mathbf{T}_0$ by $f^{(0)}(x) = x$ and $f^{(j+1)}(x) = f(f^{(j)}(x))$. We claim that

$$A(n,a) = (\mathsf{P}_0(\varepsilon^{n,a})(n)(\mathrm{id}))_0 = (\mathsf{B}(\langle\,\rangle))_0$$

where A is the Ackermann function. For instance, $A(3,a)$ is the first entry in an optimal play of a sequential game with selection functions $\varepsilon_i^{3,a}$ and id as the outcome function. We sketch its derivation below.

$$\mathsf{B}(x_0,x_1,x_2) = \varepsilon_3^{3,a}(\lambda x.\langle x_0,x_1,x_2,x,0,0,\dots\rangle),0,0,\dots = x_2+1,0,\dots.$$

$$\mathsf{B}(x_0,x_1) = \varepsilon_2^{3,a}(\lambda x.\langle x_0,x_1,x,x+1,0,\dots\rangle),\dots = (\lambda x.x+1)^{(x_1+1)}(1),\dots = x_1+2,\dots$$

$$\mathsf{B}(x_0) = \varepsilon_1^{3,a}(\lambda x.\langle x_0,x,x+2,\dots\rangle),\dots = (\lambda x.x+2)^{(x_0+1)}(1),\dots = 2x_0+3,\dots$$

$$\mathsf{B}(\langle\,\rangle) = \varepsilon_0^{3,a}(\lambda x.\langle x,2x+3,\dots\rangle),\dots = (\lambda x.2x+3)^{(a+1)}(1),\dots = 2^{(a+3)}-3,\dots.$$

Similarly, the first entry in the 5-round game with selection functions $\varepsilon_i^{4,a}$ is $(2\uparrow^2 n+3)-3$ and so on. Thus the product of selection functions over $\mathbb{N}$ allows a much higher rate of growth than primitive recursion over $\mathbb{N}$.

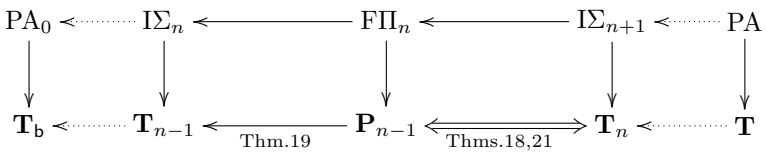


Figure 1 Fragments of Peano arithmetic and corresponding fragments of system $\mathbf{T}$.

To summarise, we have shown that the finite product of selections is equivalent to the higher type recursor in the sense that $\mathbf{P}_n \Leftrightarrow \mathbf{T}_{n+1}$ over $\mathbf{T}_0$. We conclude by pointing out that these equivalences actually hold over a much weaker base theory, and that in particular the finite product of lowest type defines the primitive recursors of lowest type over a weak fragment of $\mathbf{T}_0$. This follows from the observation that in establishing the equivalences we only make use of a very restricted class of primitive recursive functions, namely concatenation of sequences and definition by cases.

▶ **Definition 24.** 1. The fragment $\mathbf{T}_\mathsf{b} \subset \mathbf{T}_0$ of system $\mathbf{T}$ is defined to be the $\mathbf{T}_0$ but with constants for primitive recursion eliminated, save for definition by cases and concatenation $*$ for all types.
2. The binary product of selection functions can be defined in the language of $\mathbf{T}_\mathsf{b}$. Therefore, we define the theory $\tilde{\mathbf{P}}_n$ to be $\mathbf{T}_\mathsf{b}$ plus the finite product of selection functions $\mathsf{P}^{X,R}$ for all types X with $\deg(X) \leq n$.

It is easy to see that in the proof of Theorem 19 we only need to assume $\mathbf{T}_\mathsf{b}$ (rather than $\mathbf{T}_0$). Therefore we obtain the following:

▶ **Theorem 25.** $\tilde{\mathbf{P}}_0 \Rightarrow \mathbf{T}_0$, *so in particular* $\tilde{\mathbf{P}}_n$ *can be identified with* $\mathbf{P}_n$, *for all* n.

Hence all uses of $\mathbf{T}_0$ above can be replaced by $\mathbf{T}_\mathsf{b}$, and as such the finite product of selection functions $\mathsf{P}^{X,R}$ is *truly* interchangeable with Gödel's primitive recursor R^X (for all types X).

6 Final Remarks

The first two authors have studied in [2] an *unbounded product of selection functions*

$$\mathsf{P}_i^{X,R}(\varepsilon)(\psi)(q) \overset{X^{\mathbb{N}}}{=} \begin{cases} \mathbf{0}^{X^{\mathbb{N}}} & \text{if } i > \psi(q(\mathbf{0})) \\ \left(\varepsilon_i \otimes \mathsf{P}_{i+1}^{X,R}(\varepsilon)(\psi)\right)(q) & \text{if } i \leq \psi(q(\mathbf{0})) \end{cases}$$

where the fixed bound m (cf. Definition 10) is replaced by a bounding function ψ on the canonical outcome $q(\mathbf{0})$. When ψ is the constant functional m we obtain the finite product as a particular case of this. They have shown that this unbounded product is primitive recursively equivalent to Spector's bar recursion [17], and hence it is precisely what is needed to (dialectica) interpret full classical analysis. Combining this with our results above show that the iterated product of selection functions P provides a uniform link between Gödel's primitive recursor R and Spector's bar recursion, and hence, a uniform way to interpret arithmetic and analysis.

Figure 1 shows the different subsystems of Peano arithmetic we have considered, and the corresponding fragments of system $\mathbf{T}$ needed for a dialectica interpretation. Parsons

has shown that the fragment of PA based on Π_n finite choice is strictly weaker than the fragment based on Σ_{n+1} induction. Nevertheless, we have shown that the product of selection functions of type level $n-1$ (which directly interprets Π_n-FAC) is equivalent (over a weak theory) to the recursor of type n (which directly interprets Σ_{n+1}-IND).

Given that $\mathrm{F}\Pi_n$ is strictly weaker than $\mathrm{I}\Sigma_{n+1}$, we conclude with the question of whether $\mathrm{F}\Pi_n$ can be (dialectica) interpreted in a fragment of $\mathbf{T}$ that is strictly weaker than $\mathbf{P}_n$, or whether it is in fact equivalent to $\mathrm{I}\Sigma_{n+1}$ on a computational level, despite being logically weaker?

Acknowledgements. The authors are grateful to Ulrich Kohlenbach for pointing out the proof in [16] of the equivalence between Π_n-FAC and Π_n-BC.

References

1 S. R. Buss, editor. *Handbook of Proof Theory*, volume 137. Elsevier, Amsterdam, 1998.

2 M. H. Escardó and P. Oliva. Bar recursion and products of selection functions. Submitted for publication, 2010.

3 M. H. Escardó and P. Oliva. Computational interpretations of analysis via products of selection functions. In F. Ferreira, B. Lowe, E. Mayordomo, and L. M. Gomes, editors, *Computability in Europe 2010, LNCS 6158*, pages 141–150. Springer, 2010.

4 M. H. Escardó and P. Oliva. The Peirce translation and the double negation shift. In F. Ferreira, B. Löwe, E. Mayordomo, and L. M. Gomes, editors, *Programs, Proofs, Processes - CiE 2010, LNCS 6158*, pages 151–161. Springer, 2010.

5 M. H. Escardó and P. Oliva. Selection functions, bar recursion, and backward induction. *Mathematical Structures in Computer Science*, 20(2):127–168, 2010.

6 M. H. Escardó and P. Oliva. Sequential games and optimal strategies. *Royal Society Proceedings A*, 467:1519–1545, 2011.

7 K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12:280–287, 1958.

8 U. Kohlenbach. *Applied Proof Theory: Proof Interpretations and their Use in Mathematics*. Monographs in Mathematics. Springer, 2008.

9 G. Kreisel. On the interpretation of non-finitist proofs, part I. *The Journal of Symbolic Logic*, 16:241–267, 1951.

10 G. Kreisel. On the interpretation of non-finitist proofs, part II: Interpretation of number theory. *The Journal of Symbolic Logic*, 17:43–58, 1952.

11 S. Kuroda. Intuitionistische Untersuchungen der formalistischen Logik. *Nagoya Mathematical Journal*, 3:35–47, 1951.

12 H. Lessan. *Models of Arithmetic*. PhD thesis, Manchester University, 1978.

13 J. B. Paris and L. A. S. Kirby. Σ_n-collection schemas in arithmetic. In *Logic Colloquium '77*, pages 199–210. North Holland, Amsterdam, 1978.

14 C. Parsons. On a number theoretic choice schema and its relation to induction. In A. Kino, J. Myhill, and R. E. Vesley, editors, *Intuitionism and Proof Theory: Proceedings of the Summer Conference at Buffalo, N.Y. 1968*, pages 459–473. North Holland, Amsterdam, 1970.

15 C. Parsons. On n-quantifier induction. *The Journal of Symbolic Logic*, 37:466–482, 1972.

16 W. Sieg. Fragments of arithmetic. *Annals of Pure and Applied Logic*, 28:33–71, 1985.

17 C. Spector. Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles in current intuitionistic mathematics. In F. D. E. Dekker, editor, *Recursive Function Theory: Proc. Symposia in Pure Mathematics*, volume 5, pages 1–27. American Mathematical Society, Providence, Rhode Island, 1962.

Unifying Büchi Complementation Constructions

Seth Fogarty[1], Orna Kupferman[2], Moshe Y. Vardi[1], and Thomas Wilke[3]

1 **Department of Computer Science, Rice University**
2 **School of Computer Science and Engineering, Hebrew University of Jerusalem**
3 **Institut für Informatik, Christian-Albrechts-Universität zu Kiel**

Abstract

Complementation of Büchi automata, required for checking automata containment, is of major theoretical and practical interest in formal verification. We consider two recent approaches to complementation. The first is the *rank-based approach* of Kupferman and Vardi, which operates over a DAG that embodies all runs of the automaton. This approach is based on the observation that the vertices of this DAG can be ranked in a certain way, termed an *odd ranking*, iff all runs are rejecting. The second is the *slice-based approach* of Kähler and Wilke. This approach tracks levels of "split trees" – run trees in which only essential information about the history of each run is maintained. While the slice-based construction is conceptually simple, the complementing automata it generates are exponentially larger than those of the recent rank-based construction of Schewe, and it suffers from the difficulty of symbolically encoding levels of split trees.

In this work we reformulate the slice-based approach in terms of run DAGs and preorders over states. In doing so, we begin to draw parallels between the rank-based and slice-based approaches. Through deeper analysis of the slice-based approach, we strongly restrict the nondeterminism it generates. We are then able to employ the slice-based approach to provide a new odd ranking, called a *retrospective ranking*, that is different from the one provided by Kupferman and Vardi. This new ranking allows us to construct a deterministic-in-the-limit rank-based automaton with a highly restricted transition function. Further, by phrasing the slice-based approach in terms of ranks, our approach affords a simple symbolic encoding and achieves Schewe's tight bound.

1998 ACM Subject Classification F.1.1 Automata; D.2.4 Formal Methods

Keywords and phrases Büchi automata, complementation, ranks, determinism in the limit

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.248

1 Introduction

The complementation problem for nondeterministic automata is central to the automata-theoretic approach to formal verification [22]. To test that the language of an automaton $\mathcal{A}$ is contained in the language of a second automaton $\mathcal{B}$, check that the intersection of $\mathcal{A}$ with an automaton that complements $\mathcal{B}$ is empty. In model checking, the automaton $\mathcal{A}$ corresponds to the system, and the automaton $\mathcal{B}$ corresponds to a property [22]. While it is easy to complement properties given as temporal logic formulas, complementation of properties given as automata is not simple. Indeed, a word w is rejected by a nondeterministic automaton $\mathcal{A}$ if *all* runs of $\mathcal{A}$ on w reject the word. Thus, the complementary automaton

* **Acknowledgments** The authors are grateful to Yoad Lustig for his extensive help in analyzing the original slice-based construction. Work supported in part by NSF grants CCF-0728882, and CNS-1049862, by BSF grant 9800096, and by gift from Intel. Proofs and additional material available at `http://www.cs.rice.edu/~vardi/papers/csl11rj.pdf`

has to consider all possible runs, and complementation has the flavor of determinization. Representing liveness, fairness, or termination properties requires automata that recognize languages of infinite words. Most commonly considered are nondeterministic Büchi automata, in which some of the states are designated as accepting, and a run is accepting if it visits accepting states infinitely often [2]. For automata on finite words, determinization, and hence also complementation, is done via the subset construction [15]. For Büchi automata the subset construction is not sufficient, and optimal complementation constructions are more complicated [11].

Efforts to develop simple complementation constructions for Büchi automata started early in the 60s, motivated by decision problems of second-order logics. Büchi suggested a complementation construction for nondeterministic Büchi automata that involved a Ramsey-based combinatorial argument and a doubly-exponential blow-up in the state space [2]. Thus, complementing an automaton with n states resulted in an automaton with $2^{2^{O(n)}}$ states. In [19], Sistla et al. suggested an improved implementation of Büchi's construction, with only $2^{O(n^2)}$ states, which is still not optimal. Only in [16] Safra introduced a determinization construction, based on *Safra trees*, which also enabled a $2^{O(n \log n)}$ complementation construction, matching a lower bound described by Michel [11]. A careful analysis of the exact blow-up in Safra's and Michel's bounds, however, reveals an exponential gap in the constants hiding in the $O()$ notations: while the upper bound on the number of states in the complementary automaton constructed by Safra is n^{2n}, Michel's lower bound involves only an $n!$ blow up, which is roughly $(n/e)^n$. In addition, Safra's construction has been resistant to optimal implementations [1], which has to do with the complicated combinatorial structure of its states and transitions, which can not be encoded symbolically.

The use of complementation in practice has led to a resurgent interest in the exact blow-up that complementation involves and the feasibility of a symbolic complementation construction. In 2001, Kupferman and Vardi suggested a new analysis of runs of Büchi automata that led to a simpler complementation construction [10]. In this analysis, one considers a DAG that embodies all the runs of an automaton $\mathcal{A}$ on a given word w. It is shown in [10] that the nodes of this DAG can be mapped to ranks, where the rank of a node essentially indicates the progress made towards a suffix of the run with no accepting states. Further, all the runs of $\mathcal{A}$ on w are rejecting iff there is a *bounded odd ranking* of the DAG: one in which the maximal rank is bounded, ranks along paths do not increase, paths become trapped in odd ranks, and nodes associated with accepting states are not assigned an odd rank. Consequently, complementation can circumvent Safra's determinization construction along with the complicated data structure of Safra trees, and can instead be based on an automaton that guesses an odd ranking. The state space of such an automaton is based on annotating states in subsets with the guessed ranks. Beyond the fact that the *rank-based construction* can be implemented symbolically [20], it gave rise to a sequence of works improving both the blow-up it involves and its implementation in practice. The most notable improvements are the introduction of tight rankings [5] and Schewe's improved cut-point construction [17]. These improvements tightened the $(6n)^n$ upper bound of [10] to $(0.76n)^n$. Together with recent work on a tighter lower bound [23], the gap between the upper and lower bound is now a quadratic term. Addressing practical concerns, Doyen and Raskin have introduced a useful subsumption technique for the rank-based approach [4].

In an effort to unify Büchi complementation with other operations on automata, Kähler and Wilke introduced yet another analysis of runs of nondeterministic Büchi automata [7]. The analysis is based on *reduced split trees*, which are related to the Müller-Schupp trees used for determinization [13]. A reduced split tree is a binary tree whose nodes are sets of

states as follows: the root is the set of initial states; and given a node associated with a set of states, its left child is the set of successors that are accepting, while the right child is the set of successors that are not accepting. In addition, each state of the automaton appears at most once in each level of the binary tree: if it would appear in more than one set, it occurs only in the leftmost one. The construction that follows from the analysis, termed the *slice-based construction*, is simpler than Safra's determinization, but its implementation suffers from similar difficulties: the need to refer to leftmost children requires encoding of a preorder, and working with reduced split trees makes the transition relation between states awkward. Thus, as has been the case with Safra's construction, it is not clear how the slice-based approach can be implemented symbolically. This is unfortunate, as the slice-based approach does offer a very clean and intuitive analysis, suggesting that a better construction is hidden in it.

In this paper we reveal such a hidden, elegant, construction, and we do so by unifying the rank-based and the slice-based approaches. Before we turn to describe our construction, let us point to a key conceptual difference between the two approaches. This difference has made their relation of special interest and challenge. In the rank-based approach, the ranks assigned to a node bound the visits to accepting states yet to come. Thus, the ranks refer to the *future* of the run, making the rank-based approach inherently nondeterministic. In contrast, in the slice-based approach, the partition of the states of the automaton to the different sets in the tree is based on previous visits to accepting states. Thus, the partition refers to the *past* of the run, and does not depend on its future.

In order to draw parallels between the two approaches, we present a formulation of the slice-based approach in terms of run DAGs. A careful analysis of the slice-based approach then enables us to reduce the nondeterminism in the construction. We can then employ this improved slice-based approach in order to define a particular odd ranking of rejecting run DAGs, called a *retrospective ranking*. In addition to revealing the theoretical connections between the two seemingly different approaches, the new ranks lead to a complementation construction with a transition function that is smaller and deterministic in the limit: every accepting run of the automaton is eventually deterministic. This presents the first deterministic-in-the-limit complementation construction that does not use determinization. Determinism in the limit is central to verification in probabilistic settings [3] and has proven useful in experimental results [18]. Phrasing slice-based complementation as an odd ranking also immediately affords us the improved cut-point of Schewe, the subsumption operation of Doyen and Raskin, and provides an easy symbolic encoding.

2 Preliminaries

A *nondeterministic Büchi automaton on infinite words* (NBW) is a tuple $\mathcal{A}=\langle \Sigma, Q, Q^{in}, \rho, F\rangle$, where Σ is a finite alphabet, Q a finite set of states, $Q^{in} \subseteq Q$ a set of initial states, $F \subseteq Q$ a set of accepting states, and $\rho\colon Q \times \Sigma \to 2^Q$ a nondeterministic transition relation. A state $q \in Q$ is *deterministic* if for every $\sigma \in \Sigma$ it holds that $|\rho(q,\sigma)| \leq 1$. We lift the function ρ to sets R of states in the usual fashion: $\rho(R,\sigma) = \bigcup_{q\in R} \rho(q,\sigma)$.

A *run* of an NBW $\mathcal{A}$ on a word $w = \sigma_0\sigma_1 \cdots \in \Sigma^\omega$ is an infinite sequence of states $p_0, p_1, \ldots \in Q^\omega$ such that $p_0 \in Q^{in}$ and, for every $i \geq 0$, we have $p_{i+1} \in \rho(p_i, \sigma_i)$. A run is *accepting* iff $p_i \in F$ for infinitely many $i \in \mathbb{N}$. A word $w \in \Sigma^\omega$ is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. The words accepted by $\mathcal{A}$ form the *language* of $\mathcal{A}$, denoted by $L(\mathcal{A})$. The complement of $L(\mathcal{A})$, denoted $\overline{L(\mathcal{A})}$, is $\Sigma^\omega \setminus L(\mathcal{A})$. We say an automaton is *deterministic in the limit* if every state reachable from an accepting state is deterministic. Converting $\mathcal{A}$ to an equivalent deterministic in the limit automaton involves an exponential

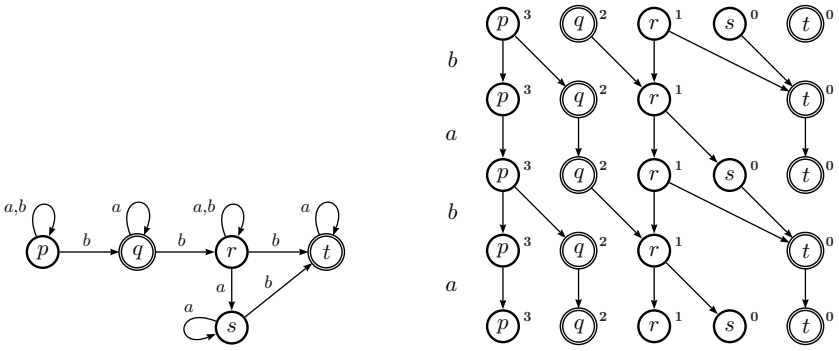


Figure 1 Left, the NBW $\mathcal{A}$, in which all states are initial. Right, the rejecting run DAG G of $\mathcal{A}$ on $w = babaabaaabaaaa\cdots$. Nodes are superscripted with the prospective ranks of Section 2.

blowup [3, 16]. One can simultaneously complement and determinize in the limit, via co-determinization into a parity automaton [14], and then converting that parity automaton to a deterministic-in-the-limit Büchi automaton, with a cost of $(n^2/e)^n$.

Consider an NBW $\mathcal{A}$ and an infinite word $w = \sigma_0\sigma_1\cdots$. The runs of $\mathcal{A}$ on w can be arranged in an infinite DAG (directed acyclic graph) $G = \langle V, E\rangle$, where

- $V \subseteq Q \times \mathbb{N}$ is such that $\langle q, i\rangle \in V$ iff some run p of $\mathcal{A}$ on w has $p_i = q$.
- $E \subseteq \bigcup_{i\geq 0} (Q\times\{i\})\times(Q\times\{i+1\})$ is s.t. $E(\langle q, i\rangle, \langle q', i{+}1\rangle)$ iff $\langle q, i\rangle \in V$ and $q' \in \rho(q, \sigma_i)$.

The DAG G, called the *run* DAG *of* $\mathcal{A}$ *on* w, embodies all possible runs of $\mathcal{A}$ on w. We are primarily concerned with *initial paths* in G: paths that start in $Q^{in} \times \{0\}$. Define a node $\langle q, i\rangle$ to be an F-node when $q \in F$, and a path in G to be *accepting* when it is both initial and contains infinitely many F-nodes. An accepting path in G corresponds to an accepting run of $\mathcal{A}$ on w. When G contains an accepting path, call G an accepting run DAG, otherwise call it a rejecting run DAG. We often consider DAGs H that are subgraphs of G. A node u is a *descendant* of v in H when u is reachable from v in H. A node v is *finite* in H if it has only finitely many descendants in H. A node v is F*-free* in H if it is not an F-node, and has no descendants in H that are F-nodes. We say a node *splits* when it has at least two children, and conversely that two nodes *join* when they share a common child.

Example 1. In Figure 1 we describe an NBW $\mathcal{A}$ that accepts words with finitely many b's. On the right is a prefix of the rejecting run DAG of $\mathcal{A}$ on $w = babaabaaabaaaa\cdots$.

If an NBW $\mathcal{A}$ does not accept a word w, then every run of $\mathcal{A}$ on w must eventually cease visiting accepting states. The notion of *ranking*s, foreshadowed in [9] and introduced in [10], uses natural numbers to track the progress of each run in the DAG towards this point. A ranking for a DAG $G = \langle V, E\rangle$ is a mapping from V to $\mathbb{N}$, in which no F-node is given an odd rank, and in which the ranks along all paths do not increase. Formally, a ranking is a function $\mathbf{r}\colon V \to \mathbb{N}$ such that if $u \in V$ is an F-node then $\mathbf{r}(u)$ is even; and for every $u, v \in V$, if $(u, v) \in E$ then $\mathbf{r}(u) \geq \mathbf{r}(v)$. Since each path starts at a finite rank and ranks cannot increase, every path eventually becomes trapped in a rank. A ranking is called an *odd ranking* if every path becomes trapped in an odd rank. Since F-nodes cannot have odd ranks, if there is an odd ranking $\mathbf{r}$, then every path in G must stop visiting accepting nodes when it becomes trapped in its final, odd, rank, and G must be a rejecting DAG.

▶ **Lemma 1.** [10] *If a run* DAG *G has an odd ranking, then G is rejecting.*

A ranking is *bounded by* l when its range is $\{0, ..., l\}$, and an NBW $\mathcal{A}$ is of rank l when for every $w \notin L(\mathcal{A})$, the rejecting DAG G has an odd ranking bounded by l. If we can prove that

an NBW $\mathcal{A}$ is of rank l, we can use the notion of odd rankings to construct a complementary automaton. This complementary NBW, denoted $\mathcal{A}^l_R$, tracks the levels of the run DAG and attempts to guess an odd ranking bounded by l. An *l-bounded level ranking* for an NBW $\mathcal{A}$ is a function $f\colon Q \to \{0,\ldots,l,\bot\}$, such that if $q \in F$ then $f(q)$ is even or $\bot$. Let $\mathcal{R}^l$ be the set of all l-bounded level rankings. The state space of $\mathcal{A}^l_R$ is based on the set of l-bounded level rankings for $\mathcal{A}$. To define transitions of $\mathcal{A}^l_R$, we need the following notion: for $\sigma \in \Sigma$ and $f, f' \in \mathcal{R}^l$, say that *f' follows f under σ* when for every $q \in Q$ and $q' \in \rho(q,\sigma)$, if $f(q) \neq \bot$ then $f'(q') \neq \bot$ and $f'(q') \leq f(q)$: i.e. no transition between f and f' on σ increases in rank. Finally, to ensure that the guessed ranking is an odd ranking, we employ the cut-point construction of Miyano and Hayashi, which maintains an obligation set of nodes along paths obliged to visit an odd rank [12]. For a level ranking f, let $even(f) = \{q \mid f(q) \text{ is even}\}$ and $odd(f) = \{q \mid f(q) \text{ is odd}\}$.

▶ **Definition 2.** For an NBW $\mathcal{A} = \langle \Sigma, Q, Q^{in}, \rho, F\rangle$ and $l \in \mathbb{N}$, define $\mathcal{A}^l_R$ to be the NBW $\langle \Sigma, \mathcal{R}^l \times 2^Q, \langle f^{in}, \emptyset\rangle, \rho_R, \mathcal{R}^l \times \{\emptyset\}\rangle$, where

- $f^{in}(q) = l$ for each $q \in Q^{in}$, $\bot$ otherwise.
- $\rho_R(\langle f, O\rangle, \sigma) = \begin{cases} \{\langle f',\ \rho(O,\sigma) \setminus odd(f')\rangle \mid f' \text{ follows } f \text{ under } \sigma\} & \text{if } O \neq \emptyset, \\ \{\langle f',\ even(f')\rangle \mid f' \text{ follows } f \text{ under } \sigma\} & \text{if } O = \emptyset. \end{cases}$

By [10], for every $l \in \mathbb{N}$, the NBW $\mathcal{A}^l_R$ accepts only words rejected by $\mathcal{A}$ — exactly all words for which there exists an odd ranking with maximal rank l. In addition, [10] proves that for every rejecting run DAG there exists a bounded odd ranking. Below we sketch the derivation of this ranking. Given a rejecting run DAG G, we inductively define a sequence of subgraphs by eliminating nodes that cannot be part of accepting runs. At odd steps we remove finite nodes, while in even steps we remove nodes that are F-free. Formally, define a sequence of subgraphs as follows:

- $G_0 = G$.
- $G_{2i+1} = G_{2i} \setminus \{v \mid v \text{ is finite in } G_{2i}\}$.
- $G_{2i+2} = G_{2i+1} \setminus \{v \mid v \text{ is } F\text{-free in } G_{2i+1}\}$.

It is shown in [6, 10] that only $m = 2|Q \setminus F|$ steps are necessary to remove all nodes from a rejecting run DAG: G_m is empty. Nodes can be ranked by the last graph in which they appear: for every node $u \in G$, the *prospective rank* of u is the index i such that $u \in G_i$ but $u \notin G_{i+1}$. The *prospective ranking* of G assigns every node its prospective rank. Paths through G cannot increase in prospective rank, and no F-node can be given an odd rank: thus the prospective ranking abides by the requirements for rankings. We call these rankings prospective because the rank of a node depends solely on its descendants. By [10], if G is a rejecting run DAG, then the prospective ranking of G is an odd ranking bounded by m. By the above, we thus have the following.

▶ **Theorem 3.** [10] *For every NBW $\mathcal{A}$, it holds that $L(\mathcal{A}^m_R) = \overline{L(\mathcal{A})}$.*

Example 2. In Figure 1, nodes for states s and t are finite in G_0. Without these nodes, r-nodes are F-free in G_1. Similarly, q-nodes are finite in G_2, and p-nodes are F-free in G_3.

Karmarkar and Chakraborty derive both theoretical and practical benefits from exploiting properties of this prospective ranking: they demonstrated an unambiguous complementary automaton that, for certain classes of problems, is exponentially smaller than $\mathcal{A}^m_R$ [8].

Tight Rankings: For an odd ranking $\mathbf{r}$ and $l \in \mathbb{N}$, let $max\_rank(\mathbf{r}, l)$ be the maximum rank that $\mathbf{r}$ assigns a vertex on level l of the run DAG. We say that $\mathbf{r}$ is *tight* [1] if there exists

[1] This definition of tightness is weaker that of [5], but does not affect the resulting bounds.

an $i \in \mathbb{N}$ such that, for every level $l \geq i$, all odd ranks below $max\_rank(\mathbf{r}, l)$ appear on level l. It is shown in [5] that the retrospective ranking is tight. This observation suggests two improvements to $\mathcal{A}^m_R$. First, we can postpone, in an unbounded manner, the level in which it starts to guess the level ranking. Until this point, $\mathcal{A}^m_R$ may use sets of states to deterministically track only the levels of the run DAG, with no attempt to guess the ranks. Second, after this point, $\mathcal{A}^m_R$ can restrict attention to *tight level rankings* – ones in which all the odd ranks below the maximal rank appear. Formally, say a level ranking f with a maximum rank $mr = \max\{f(q) \mid q \in Q,\ f(q) \neq \bot\}$ is tight when, for every odd $i \leq mr$, there exists a $q \in Q$ such that $f(q) = i$. Let $\mathcal{R}^m_T$ be the subset of $\mathcal{R}^m$ that contains only tight level rankings. The size of $\mathcal{R}^m_T$ is at most $(0.76n)^n$ [5]. Including the cost of the cut-point construction, this reduces the state space of $\mathcal{A}^m_R$ to $(0.96n)^n$.

3 Analyzing DAGs With Profiles

In this section we present an alternate formulation of the slice-based complementation construction of Kähler and Wilke [7]. Whereas Kähler and Wilke approached the problem through reduced split trees, we derive the slice-based construction directly from an analysis of the run DAG. This analysis proceeds by pruning G in two steps: the first removes edges, and the second removes vertices.

Profiles: Consider a run DAG $G = \langle V, E\rangle$. Let $l\colon V \to \{0,1\}$ be such that $l(\langle q, i\rangle) = 1$ if $q \in F$ and $l(\langle q, i\rangle) = 0$ otherwise. Thus, l labels F-nodes by 1 and all other nodes by 0. The *profile* of a path in G is the sequence of labels of nodes in the path. The profile of a node is then the lexicographically maximal profile of all initial paths to that node. Formally, let $\leq$ be the lexicographic ordering on $\{0,1\}^* \cup \{0,1\}^\omega$. The profile of a finite path $b = v_0, v_1, \ldots, v_n$ in G, written h_b, is $l(v_0)l(v_1)\cdots l(v_n)$, and the profile of an infinite path $b = v_0, v_1, \ldots$ is $h_b = l(v_0)l(v_1)\cdots$. Finally, the profile of a node v, written h_v, is the lexicographically maximal element of $\{h_b \mid b \text{ is an initial path to } v\}$. The lexicographic order of profiles induces a preorder over nodes.

We define the sequence of preorders $\preceq_i$ over the nodes on each level of the run DAG as follows. For every two nodes u and v on a level i, we have that $u \prec_i v$ if $h_u < h_v$, and $u \approx_i v$ if $h_u = h_v$. For convenience, we conflate nodes on the ith level of the run DAG with their states when employing this preorder, and say $q \preceq_i r$ when $\langle q, i\rangle \preceq_i \langle r, i\rangle$. Note that $\approx_i$ is an equivalence relation. Since the final element of a node's profile is 1 iff the node is an F-node, all nodes in an equivalence class must agree on membership in F. We call an equivalence class an F-class when all its members are F-nodes, and a non-F-class when none of its members is an F-node. We now use profiles in order to remove from G edges that are not on lexicographically maximal paths. Let G' be the subgraph of G obtained by removing all edges $\langle u, v\rangle$ for which there is another edge $\langle u', v\rangle$ such that $u \prec_{|u|} u'$. Formally, $G' = \langle V, E'\rangle$ where $E' = E \setminus \{\langle u, v\rangle \mid \text{there exists } u' \in V \text{ such that } \langle u', v\rangle \in E \text{ and } u \prec_{|u|} u'\}$.

▶ **Lemma 4.** *For every two nodes u and v, if $(u, v) \in E'$, then $h_v \in \{h_u 0, h_u 1\}$.*

Proof. Assume by way of contradiction that $h_v \notin \{h_u 0, h_u 1\}$. Recall that h_v is the lexicographically maximal element of $\{h_b \mid b \text{ is an initial path to } v\}$. Thus our assumption entails an initial path b to v so that $h_b > h_u 1$. Let u' be $b_{|u|}$: the node on the same level of G as u. Since b is a path to v, it holds that $(u', v) \in E$. Further, since $h_b > h_u 1$, it must be that $h_{u'} > h_u$. By definition of E', the presence of (u', v) where $h_{u'} > h_u$ precludes the edge (u, v) from being in E' — a contradiction. ◀

Note that while it is possible for two nodes with different profiles to share a child in G, Lemma 4 precludes this possibility in G'. If two nodes join in G', they must have the same profile and be in the same equivalence class. We can thus conflate nodes and equivalence classes, and for every edge $(u, v) \in E'$, consider $[v]$ to be the child of $[u]$. Lemma 4 then entails that the class $[u]$ can have at most two children: the class of F-nodes with profile h_u1, and the class of non-F-nodes with profile h_u0. We call the first class the F-child of $[u]$, and the second class the non-F-child of $[u]$.

By using lexicographic ordering we can derive the preorder for each level $i+1$ of the run DAG solely from the preorder for the previous level i. To determine the relation between two nodes, we need only know the relation between the parents of those nodes, and whether the nodes are F-nodes. Formally, we have the following.

▶ **Lemma 5.** *For all nodes u, v on level i, and nodes u', v' where $E'(u, u')$ and $E'(v, v')$:*
- *If $u \prec_i v$, then $u' \prec_{i+1} v'$.*
- *If $u \approx_i v$ and either both u' and v' are F-nodes, or neither are F-nodes, then $u' \approx_{i+1} v'$.*
- *If $u \approx_i v$ and v' is an F-node while u' is not, then $u' \prec_{i+1} v'$.*

Proof. If $u \prec_i v$, then $h_u < h_v$ and, by Lemma 4, we know that $h_{u'} \in \{h_u0, h_u1\}$ must be smaller than $h_{v'} \in \{h_v0, h_v1\}$, implying that $u' \prec_{i+1} v'$. If $u \approx_i v$, we have three sub-cases. If v' is an F-node and u' is not, then $h_{u'} = h_u0 = h_v0 < h_v1 = h_{v'}$, and $u' \prec_{i+1} v'$. If both u' and v' are F-nodes, then $h_{u'} = h_u1 = h_v1 = h_{v'}$, and $u' \approx_i v'$. Finally, if neither are F-nodes, then $h_{u'} = h_u0 = h_v0 = h_{v'}$ and $u' \approx_i v'$. ◀

We now demonstrate that by keeping only edges associated with lexicographically maximal profiles, G' captures an accepting path from G.

▶ **Lemma 6.** *G' has an accepting path iff G has an accepting path.*

Proof. In one direction, if G' has an accepting path, then its superset G has the same path.

In the other direction, assume G has an accepting path. Consider the set P of accepting paths in G. We prove that there is a lexicographically maximal element $\pi \in P$. To begin, we construct an infinite sequence, $P_0, P_1, \ldots$, of subsets of P such that the elements of P_i are lexicographically maximal in the first $i+1$ positions. If P contains paths starting in an F-node, then $P_0 = \{b \mid b \in P,\ b_0 \text{ is an } F\text{-node}\}$ is all elements beginning in F-nodes. Otherwise $P_0 = P$. Inductively, if P_i contains an element b such that b_{i+1} is an F-node, then $P_{i+1} = \{b \mid b \in P_i,\ b_{i+1} \text{ is an } F\text{-node}\}$. Otherwise $P_{i+1} = P_i$. For convenience, define the predecessor of P_i to be P if $i = 0$, and P_{i-1} otherwise. Note that since G has an accepting path, P is non-empty. Further, every set P_i is not equal to its predecessor P' only when there is a path in P' with an F-node in the ith position. In this case, that path is in P_i. Thus every P_i is non-empty.

First, we prove that there is a path $\pi \in \bigcap_{i \geq 0} P_i$. Consider the sequence $U_0, U_1, U_2, \ldots$ where U_i is the set of nodes that occur at position i in runs in P_i. Formally, $U_i = \{u \mid u \in G,\ b \in P_i,\ u = b_i\}$. Each node in U_{i+1} has a parent in U_i, although it may not have a child in U_{i+2}. We can thus connect the nodes in $\bigcup_{i>0} U_i$ to their parents, forming a sub-DAG of G. As every P_i is non-empty, every U_i is non-empty, and this DAG has infinitely many nodes. Since each node has at most n children, by Koïig's Lemma there is an initial path π through this DAG, and thus through G. We now show by induction that $\pi \in P_i$ for every i. As a base case, $\pi \in P$. Inductively, assume π is in the predecessor P' of P_i. The set P_i is either P', in which case $\pi \in P_i$, or the set $\{b \mid b \in P',\ b_i \text{ is an } F\text{-node}\}$. In this latter case, as U_i consists only of F-nodes, the node π_i must be an F-node. and $\pi \in P_i$.

Second, having established that there must be an element $\pi \in \bigcap_{i \geq 0} P_i$, we prove π is lexicographically maximal in P. Assume by way of contradiction that there exists an accepting path π' so that $h_{\pi'} > h_\pi$. Let k be the first point where $h_{\pi'}$ differs from h_π. At this point, it must be that π_k is not an F node, while π'_k is an F node. However, π' is an accepting path that shares a profile with π up until this point. As π is in the predecessor P' of P_k, it must also be that π' is in P'. By definition, P_k then would be $\{b \mid b \in P',\ b_k \text{ is an } F\text{-node}\}$. This would imply $\pi \notin P_k$, a contradiction.

Finally, we demonstrate that every edge in π occurs in G'. Assume by way of contradiction that some edge (π_i, π_{i+1}) is in E but not in E'. This implies there is a node u on level i such that (u, π_{i+1}) is in E and $\pi_i \prec_i u$. Since $u \in G$, there is an initial path b to u. Thus, the path $b, u, \pi_{i+1}, \pi_{i+2} \ldots$ is an accepting path in G. This path is lexicographically larger than π, contradicting the second claim above. Hence, π is an accepting path in G'. ◀

In the next stage, we remove from G' finite nodes. Let $G'' = G' \setminus \{v \mid v \text{ is finite in } G'\}$. Note there may be nodes that are not finite in G, but are finite in G'. It is not hard to see that G may have infinitely many F-nodes and still not contain a path with infinitely many F-nodes. Indeed, G may have infinitely many paths each with finitely many F-nodes. We now show that the transition from G via G' to G'' removes this possibility, and the presence of infinitely many F-nodes in G'' does imply a path with infinitely many F-nodes.

▶ **Lemma 7.** *G has an accepting path iff G'' has infinitely many F-nodes.*

Proof. If G has an accepting path, then by Lemma 6 G' contains an accepting path. Every node in this path is infinite in G', and thus this path is preserved in G''. This path contains infinitely many F-nodes, and thus G'' contains infinitely many F-nodes.

In the other direction, we consider the DAG over equivalence classes induced by G''. Given a node u in G'', recall that its equivalence class in G'' contains all states v such that $v \in G''$ and $h_u = h_v$. Given two equivalence classes U and V, recall that V is a child of U when there are $u \in U$, $v \in V$, and $E''(u, v)$. As mentioned above, once we have pruned edges not in G', two nodes of different classes cannot join. Thus this DAG is a tree. Further, as every node u in G'' is infinite and has a child, its equivalence class must also have a child. Thus the DAG of classes in G'' is a leafless tree. The width of this tree must monotonically increase and is bounded by n. It follows that at some level j the tree reaches a stable width. We call this level j the *stabilization level* of G.

After the stabilization level, each class U has exactly one child: as noted above, U cannot have zero children, and if U had two children the width of the tree would increase. Therefore, we identify each equivalence class on level j of G'' with its unique branch of children in G'', which we term its *pipe*. These pipes form a partition of nodes in G'' after j. Every node in these pipes has an ancestor, or it would not be in the DAG, and has a child, or it would not be infinite and in G''. Therefore each node is part of an infinite path in this pipe. Thus, the pipe with infinitely many F-classes contains only accepting paths. These paths are accepting in G, which subsumes G''. ◀

In the proof above we demonstrated there is a *stabilization level* j at which the number of equivalence classes in G'' stabilized, and discussed the *pipes* of G'': the single chain of descendants from each equivalence class on the stabilization level j of G''.

Example 3. Figure 2 displays G'' for the example of Figure 1. Edges removed from G' are dotted: at levels 1 and 3. When both r and s transition to t, they have the same profile and both edges remain. All but the first q-node are finite in G'. The stabilization level is 0.

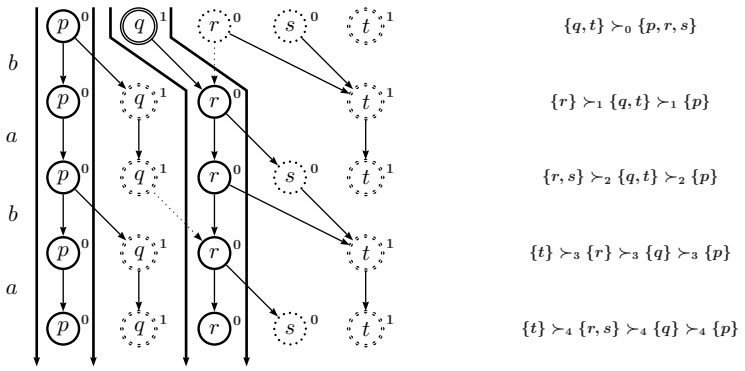


Figure 2 The run DAG G'', where dotted edges were removed from G and dotted states were removed from G'. Nodes are superscripted with their l-labels. Bold lines denote the pipes of G''. The lexicographic order of equivalence classes for each level of G' is to the right.

Complementing With Profiles: We now complement $\mathcal{A}$ by constructing an NBW, $\mathcal{A}_S$, that employs Lemma 7 to determine if a word is in $L(\mathcal{A})$. This construction is a reformulation of the slice-based approach of [7] in the framework of run DAGs. The NBW $\mathcal{A}_S$ tracks the levels of G' and guesses which nodes are finite in G' and therefore do not occur in G''. To track G', the automaton $\mathcal{A}_S$ stores at each point in time a set S of states that occurs on each level. The sets S are labeled with a guess of which nodes are finite and which are infinite. States that are guessed to be infinite, and thus correspond to nodes in G'', are labeled $\top$, and states that are guessed to be finite, and omitted from G'', are labeled $\bot$. In order to track the edges of G', and thus maintain this labeling, $\mathcal{A}_S$ needs to know the lexicographic order of nodes. Thus $\mathcal{A}_S$ also maintains the preorder $\preceq_i$ over states on the corresponding level of the run DAG. To enforce that states labeled $\bot$ are indeed finite, $\mathcal{A}_S$ employs the cut-point construction of Miyano and Hayashi [12], keeping an "obligation set" of states currently being verified as finite. Finally, to ensure the word is rejected, $\mathcal{A}_S$ must enforce that there are finitely many F-nodes in G''. To do so, S_A uses a bit b to guess the level from which no more F-nodes appear in G''. After this point, all F-nodes must be labeled $\bot$.

Before we define $\mathcal{A}_S$, we formalize *preordered subsets* and operations over them. For a set Q of states, define $\mathbf{Q} = \{\langle S, \preceq\rangle \mid S \subseteq Q \text{ and } \preceq \text{ is a preorder over } S\}$ to be the set of preordered subsets of Q. Let $\langle S, \preceq\rangle$ be an element in $\mathbf{Q}$. When considering the successors of a state, we want to consider edges that remain in G'. For every state $q \in S$ and $\sigma \in \Sigma$, define $\rho_{\langle S,\preceq\rangle}(q,\sigma) = \{r \in \rho(q,\sigma) \mid \text{for every } q' \in S, \text{ if } r \in \rho(q',\sigma) \text{ then } q' \preceq q\}$. Now define the *$\sigma$-successor of* $\langle S, \preceq\rangle$ as the tuple $\langle \rho(S,\sigma), \preceq'\rangle$, where for every $q, r \in S$, $q' \in \rho_{\langle S,\preceq\rangle}(q,\sigma)$, and $r' \in \rho_{\langle S,\preceq\rangle}(r,\sigma)$:

- If $q \prec r$, then $q' \prec' r'$
- If $q \approx r$ and either both $r' \in F$ and $q' \in F$, or both $r' \notin F$ and $q' \notin F$, then $q' \approx' r'$.
- If $q \approx r$ and one of q' and r', say r', is in F while the other, q', is not, then $q' \prec' r'$.

We now define $\mathcal{A}_S$. The states of $\mathcal{A}_S$ are tuples $\langle S, \preceq, \lambda, O, b\rangle$ where: $\langle S, \preceq\rangle \in \mathbf{Q}$ is preordered subset of Q; $\lambda\colon S \to \{\top, \bot\}$ is a labeling indicating which states are guessed to be finite ($\bot$) or infinite ($\top$), $O \subseteq S$ is the obligation set, and $b \in \{0,1\}$ is a bit indicating whether we have seen the last F-node in G''. To transition between states of $\mathcal{A}_s$, say that $\boldsymbol{t}' = \langle S', \preceq', \lambda', O', b'\rangle$ *follows* $\boldsymbol{t} = \langle S, \preceq, \lambda, O, b\rangle$ *under* σ when:

(1) $\langle S', \preceq'\rangle$ is the σ-successor of $\langle S, \preceq\rangle$.
(2) λ' is such that for every $q \in S$:
   - If $\lambda(q) = \top$, then there exists $r \in \rho_{\langle S,\preceq\rangle}(q,\sigma)$ such that $\lambda'(r) = \top$,

- If $\lambda(q) = \bot$, then for every $r \in \rho_{\langle S,\preceq\rangle}(q,\sigma)$, it holds that $\lambda'(r) = \bot$.

(3) $O' = \begin{cases} \bigcup_{q\in O} \rho_{\langle S,\preceq\rangle}(q,\sigma) & O \neq \emptyset, \\ \{q \mid q \in S' \text{ and } \lambda'(q) = \bot\} & O = \emptyset. \end{cases}$

(4) $b' \geq b$.

We want to ensure that runs of $\mathcal{A}_S$ reach a suffix where all F-nodes are labeled finite. Given a state of $\mathcal{A}_S$ $\langle S, \preceq, \lambda, O, b\rangle$, say that λ is *F-free* if for every $q \in S \cap F$ we have $\lambda(q) = \bot$.

▶ **Definition 8.** For an NBW $\mathcal{A} = \langle \Sigma, Q, Q^{in}, \rho, F\rangle$, let $\mathcal{A}_S$ be the NBW $\langle \Sigma, Q_S, Q^{in}_S, \rho_S, F_S\rangle$, where:

- $Q_S = \{\langle S, \preceq, \lambda, O, b\rangle \mid \text{if } b = 1 \text{ then } \lambda \text{ is } F\text{-free}\}$,
- $Q^{in}_S = \{\langle Q^{in}, \preceq, \lambda, \emptyset, 0\rangle \mid \text{for all } q, r \in Q^{in},\ q \preceq r \text{ iff } q \notin F \text{ or } r \in F\}$,
- $\rho_S(\mathbf{t}, \sigma) = \{\mathbf{t}' \mid \mathbf{t}' \text{ follows } \mathbf{t} \text{ under } \sigma\}$, and
- $F_S = \{\langle S, \preceq, \lambda, \emptyset, 1\rangle\}$.

▶ **Theorem 9.** *For every NBW $\mathcal{A}$, it holds that $L(\mathcal{A}_S) = \overline{L(\mathcal{A})}$.*

The proof of correctness for Theorem 9 is straightforward and based on correlating runs of $\mathcal{A}_S$ with G and its subgraphs. If $n = |Q|$, the number of preordered subsets is roughly $(0.53n)^n$ [21]. As there are 2^n labelings, and a further 2^n obligation sets, the state space of $\mathcal{A}_s$ is at most $(2n)^n$. The slice-based automaton obtained in [7] coincides with $\mathcal{A}_S$, modulo the details of labeling states and the cut-point construction. Whereas the correctness proof in [7] is given by means of reduced split trees, here we proceed directly on the run DAG.

4 Retrospection

Consider an NBW $\mathcal{A}$. So far, we presented two complementation constructions for $\mathcal{A}$, generating the NBWs $\mathcal{A}^m_R$ and $\mathcal{A}_S$. In this section we present a third construction, generating an NBW that combines the benefits of the two constructions above. Both constructions refer to the run DAG of $\mathcal{A}$. In the rank-based approach applied in $\mathcal{A}^m_R$, the ranks assigned to a node bound the visits in accepting states yet to come. Thus, the ranks refer to the future, making $\mathcal{A}^m_R$ inherently nondeterministic. On the other hand, the NBW $\mathcal{A}_S$ refers to both the past, using profiles to prune edges from G, as well as to the future, by keeping in G'' only nodes that are infinite in G'. Guessing which nodes are infinite and labeling them $\top$ inherently introduces nondeterminism into the automaton.

Our first goal in the combined construction is to reduce this latter nondeterminism. Recall that a labeling is F-free if all the states in F are labeled $\bot$. Observe that the fewer labels of $\bot$ (finite nodes) we have, the more difficult it is for a labeling to be F-free and, consequently, the more difficult it is for a run of $\mathcal{A}_S$ to proceed to the F-free suffix in which $b = 1$. It is therefore safe for $\mathcal{A}_S$ to underestimate which nodes to label $\bot$, as long as the requirement to reach an F-free suffix is maintained. We use this observation in order to introduce a purely retrospective construction.

For a run DAG G, say that a level k is an *F-finite level* of G when all F-nodes *after* level k (i.e. on a level k' where $k' > k$) are finite in G'. By Lemma 7, G is rejecting iff there is a level after which G'' has no F-nodes. As finite nodes in G' are removed from G'', we have:

▶ **Corollary 10.** *A run DAG G is rejecting iff it has an F-finite level.*

Retrospective Labeling: The labeling function λ used in the construction of $\mathcal{A}_S$ labels nodes by $\{\top, \bot\}$, with $\bot$ standing for "finite" and $\top$ standing for "infinite". In this section we introduce a variant of λ that again maps nodes to $\{\top, \bot\}$ except that now $\top$ stands for

"unrestricted", allowing us to underestimate which nodes to label $\bot$. To capture the relaxed requirements on labelings, say that a labeling λ is *legal* when every $\bot$-labeled node is finite in G'. This enables the automaton to track the labeling and its effect on F-nodes only after it guesses that an F-finite level k has been reached: all nodes *at or before* level k (i.e. on a level k' where $k' \leq k$) are unrestricted, whereas F-nodes after level k and their descendants are required to be finite. The only nondeterminism in the automaton lies in guessing when the F-finite level has been reached. This reduces the branching degree of the automaton to 2, and renders it deterministic in the limit.

The suggested new labeling is parametrized by the F-finite level k. The labeling λ^k is defined inductively over the levels of G. Let S_i be the set of nodes on level i of G. For $i \geq 0$, the function $\lambda^k \colon S_i \to \{\top, \bot\}$ is defined as follows:

- If $i \leq k$, then for every $u \in S_i$ we define $\lambda^k(u) = \top$.
- If $i > k$, then for every $u \in S_i$:
  - If u is an F-node, then $\lambda^k(u) = \bot$.
  - Otherwise, $\lambda^k(u) = \lambda^k(v)$, for a node v where $E'(v, u)$.

For λ^k to be well defined when $i > k$ and u is not an F-node, we need to show that $\lambda^k(u)$ does not depend on the choice of the node v where $E'(v, u)$ holds. By Lemma 4, all parents of a node in G' belong to the same equivalence class. Therefore, it suffices to prove that all nodes in the same class share a label: for all nodes u and u', if $u' \approx_{|u|} u$ then $\lambda^k(u) = \lambda^k(u')$. The proof proceeds by an induction on $i = |u|$. Consider two nodes u and u' on level i where $u' \approx_i u$. As a base case, if $i \leq k$, then u and u' are labeled $\top$. For $i > k$, if u is an F-node, then u' is also an F-node and $\lambda^k(u) = \lambda^k(u') = \bot$. Finally, if u and u' are both non-F-nodes, recall that all parents of u are in the same equivalence class V. As $u \approx_i u'$, Lemma 4 implies that all parents of u' are also in V By the induction hypothesis, all nodes in V share a label, and thus $\lambda^k(u) = \lambda^k(u')$.

▶ **Lemma 11.** *For a run* DAG *G and $k \in \mathbb{N}$, the labeling λ^k is legal iff k is an F-finite level for G.*

Proof. If λ^k is legal, then every $\bot$-labeled node is finite in G'. Every F-node after level k is labeled $\bot$, and thus k is an F-finite level for G. If λ^k is not legal, then there is a $\bot$-labeled node u that is infinite in G'. Every ancestor of u is also infinite. Let u' be the earliest ancestor of u (possibly u itself) so that $\lambda^k(u') = \bot$. Observe that only nodes after level k can be $\bot$-labeled, and so u' is on a level $i > k$. It must be that u' is an F-node: otherwise it would inherit its parents' label, and by assumption the parents of u' are $\top$-labeled. Thus, u' is an F-node after level k that is infinite in G', and k is not an F-finite level for G. ◀

▶ **Corollary 12.** *A run* DAG *G is rejecting iff, for some k, the labeling λ^k is legal.*

From Labelings to Rankings: In this section we derive an odd ranking for G from the function λ^k, thus unifying the retrospective analysis behind λ^k with the rank-based analysis of [10]. Consider again the DAG G' and the function λ^k. Recall that every equivalence class U has at most two child equivalence classes, one F-class and one non-F-class. Past the F-finite level k, only non-F-classes can be labeled $\top$. Hence, after level k, every $\top$-labeled equivalence class U can only have a one child that is $\top$-labeled. For every class U on level k, we consider this possibly infinite sequence of $\top$-labeled non-F-children. The odd ranking we are going to define, termed the *retrospective ranking*, gives these sequences of $\top$-labeled children odd ranks. The $\bot$-labeled classes, which lie between these sequences of $\top$-labeled classes, are assigned even ranks. The ranks increase in inverse lexicographic order, i.e. the maximal $\top$-labeled class in a level is given rank 1. As with λ^k, the retrospective ranking is

parametrized by k. The primary insight that allows this ranking is that there is no need to distinguish between two adjacent $\bot$-labeled classes. Formally, we have the following.

▶ **Definition 13** (k-retrospective ranking)**.** Consider a run DAG G, $k \in \mathbb{N}$, and a labeling $\lambda^k \colon G \to \{\top, \bot\}$. Let $m = 2|Q \setminus F|$. For a node u on level i of G, let $\alpha(u)$ be the number of $\top$-labeled classes lexicographically larger than u; $\alpha(u) = |\{[v] \mid \lambda^k(v) = \top \text{ and } u \prec_i v\}|$. The *$k$-retrospective ranking* of G' is the function $\mathbf{r}^k \colon V \to \{0..m\}$ defined for every node u on level i as follows.

$$\mathbf{r}^k(u) = \begin{cases} m & \text{if } i \leq k, \\ 2\alpha(u) & \text{if } i > k \text{ and } \lambda^k(u) = \bot, \\ 2\alpha(u) + 1 & \text{if } i > k \text{ and } \lambda^k(u) = \top. \end{cases}$$

Note that $\mathbf{r}^k$ is tight. As defined in Section 2, a ranking is tight if there exists an $i \in \mathbb{N}$ such that, for every level $l \geq i$, all odd ranks below $max\_rank(\mathbf{r}, l)$ appear on level l. For $\mathbf{r}^k$ this level is $k+1$, after which each $\top$-labeled class is given the odd rank greater by two than the rank of the next lexicographically larger $\top$-labeled class.

▶ **Lemma 14.** *For every $k \in \mathbb{N}$, the following hold:*
(1) If $u \prec_{|u|} u'$ then $\mathbf{r}^k(u) \geq \mathbf{r}^k(u')$.
(2) If $(u, v) \in E'$, then $\mathbf{r}^k(u) \geq \mathbf{r}^k(v)$.

Proof. As both claims are trivial when u is at or before level k, assume u is on level $i > k$. To prove the first claim, note that $\alpha(u) \geq \alpha(u')$: every class, $\top$-labeled or not, that is larger than u' must also be larger than u. If $\alpha(u) > \alpha(u')$, then (1) follows immediately. Otherwise $\alpha(u) = \alpha(u')$, which implies that $\lambda^k(u') = \bot$: otherwise $[u']$ would be a $\top$-labeled equivalence class larger than u, but not larger than itself. Thus $\mathbf{r}^k(u') = 2\alpha(u)$, and $\mathbf{r}^k(u) \in \{2\alpha(u),\ 2\alpha(u)+1\}$ is at least $\mathbf{r}^k(u')$.

As a step towards proving the second claim, we show that $\alpha(u) \geq \alpha(v)$. Consider every $\top$-labeled class $[v']$ where $v \prec_{i+1} v'$. The class $[v']$ must have a $\top$-labeled parent $[u']$. Since $v \prec_{i+1} v'$, the contrapositive of Lemma 5, part 1, entails that $u \preceq_i u'$. By the definition of λ^k, the class $[u']$ can only have one $\top$-labeled child class: $[v']$. We have thus established that for every $\top$-labeled class larger than v, there is a unique $\top$-labeled class larger than u, and can conclude that $\alpha(u) \geq \alpha(v)$. We now show by contradiction that $\mathbf{r}^k(u) \geq \mathbf{r}^k(v)$. For $\mathbf{r}^k(u) < \mathbf{r}^k(v)$, it must be that $\alpha(u) = \alpha(v)$, that $\mathbf{r}^k(u) = 2\alpha(u)$, and that $\mathbf{r}^k(v) = 2\alpha(u)+1$. In this case, $\lambda^k(u) = \bot$ and $\lambda^k(v) = \top$. Since a $\bot$-labeled node cannot have a $\top$-labeled child in G', this is impossible. ◀

When k is an F-finite level of G, the k-retrospective ranking is a bounded odd ranking.

▶ **Lemma 15.** *For a run DAG G and $k \in \mathbb{N}$, the function $\mathbf{r}^k$ is a ranking bounded by m. Further, if the labeling λ^k is legal then $\mathbf{r}^k$ is an odd ranking.*

Proof. There are three requirements for $\mathbf{r}^k$ to be a ranking bounded by m:

(1) *Every F-node must have an even rank.* At or before level k, every node has even rank m. After k only $\top$-labeled nodes are given odd ranks, and every F-node is labeled $\bot$.
(2) *For every $(u, v) \in E$, it must hold that $\mathbf{r}^k(u) \geq \mathbf{r}^k(v)$.* If u is at or before level k, then it has the maximal rank of m. If u is after level k, we consider two cases: edges in E', and edges in $E \setminus E'$. For edges in E', this follows from Lemma 14 (2). For edges $(u, v) \in E \setminus E'$, we know there exists a u' where $u \prec_{|u|} u'$ and $(u', v) \in E'$. By Lemma 14, $\mathbf{r}^k(u) \geq \mathbf{r}^k(u') \geq \mathbf{r}^k(v)$.

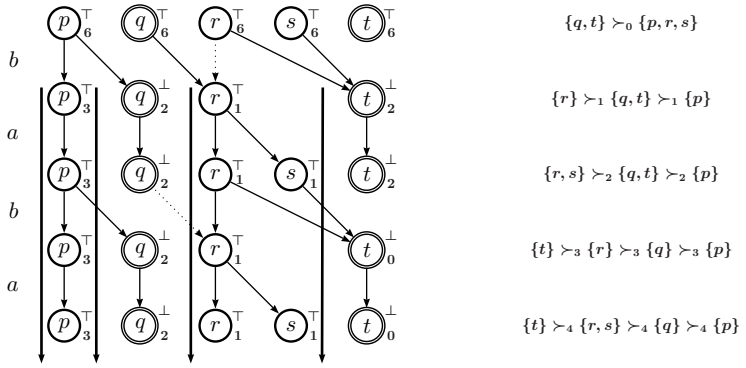


Figure 3 The run DAG G', where 0 is an F-finite level. The labels of λ^0 and ranks in $\mathbf{r}^0$ are displayed as superscripts and subscripts, respectively. The bold lines display the sequences of $\top$-labeled classes in G'. The lexicographic order of states is repeated on the right.

(3) *The rank is bounded by m.* No F-node can be $\top$-labeled. Thus the maximum number of $\top$-labeled classes on every level is $|Q \setminus F|$. The largest possible rank is given to a node smaller than all $\top$-labeled classes, which must be be a F-node and $\bot$-labeled. Thus this node is given a rank of at most $m = 2|Q \setminus F|$.

It remains to show that if λ^k is legal, then $\mathbf{r}^k$ is an odd ranking. Consider an infinite path $u_0, u_1, \ldots$ in G. We demonstrate that for every $i > k$ such that $\mathbf{r}^k(u_i)$ is an even rank e, there exists $i' > i$ such that $\mathbf{r}^k(u_{i'}) \neq e$. Since a path cannot increase in rank, this implies $\mathbf{r}^k(u_{i'}) < e$. To do so, define the sequence $U_i, U_{i+1}, \ldots$, of sets of nodes inductively as follows. Let $U_i = \{v \mid \mathbf{r}^k(v) = e\}$. For every $j \geq i$, let $U_{j+1} = \{v \mid v' \in U_j,\ (v', v) \in E'\}$. As $\mathbf{r}^k(v)$ is even only when $\lambda^k(v) = \bot$, if λ^k is legal then every node given an even rank (such as e) must be finite in G'. Therefore every element of U_i is finite in G', and thus at some $i' > i$, the set $U_{i'}$ is empty. Since $U_{i'}$ is empty, to establish that $\mathbf{r}^k(u_{i'}) \neq e$, it is sufficient to prove that for every j, if $\mathbf{r}^k(u_j) = e$, then $u_j \in U_j$.

To show that $\mathbf{r}^k(u_j) = e$ entails $u_j \in U_j$, we prove a stronger claim: for every $j \geq i$ and v on level j, if $u_j \preceq_j v$ and $\mathbf{r}^k(v) = e$, then $v \in U_j$. We proceed by induction over j. For the base case of $j = i$, this follows from the definition of U_i. For the inductive step, take a node v on level $j+1$ where $\mathbf{r}^k(v) = e$ and $u_{j+1} \preceq_{j+1} v$. We consider two cases. If $\mathbf{r}^k(u_{j+1}) \neq e$ then the path from u_i to u_{j+1} entails that $\mathbf{r}^k(u_{j+1}) < e$, and this case of the subclaim follows from Lemma 14 (1). Otherwise, it holds that $\mathbf{r}^k(u_{j+1}) = e$, and thus $\mathbf{r}^k(u_j) = e$. Let u' and v' be nodes on level j so that $(u', u_{j+1}) \in E'$ and $(v', v) \in E'$. As $u_{j+1} \preceq_{j+1} v$, the contrapositive of Lemma 5, part 1, entails that $u' \preceq_j v'$. Further, since $(u', u_{j+1}) \in E'$ and $(u_j, u_{j+1}) \in E$, we know $u_j \preceq_j u'$. By transitivity we can thus conclude that $u_j \preceq_j v'$, which along with Lemma 14 (1) entails $\mathbf{r}^k(u') = e \geq \mathbf{r}^k(v')$. As $(v', v) \in E$, Lemma 14 (2) entails that $\mathbf{r}^k(v') \geq \mathbf{r}^k(v) = e$. Thus $\mathbf{r}^k(v') = e$, and by the inductive hypothesis $v' \in U_j$. As $E'(v', v)$ holds, by definition $v \in U_{j+1}$, and our subclaim is proven. ◀

The ranking of Definition 13 is termed *retrospective* as it relies on the relative lexicographic order of equivalence classes; this order is determined purely by the history of nodes in the run DAG, not by looking forward to see which descendants are infinite or F-free in some subgraph of G.

Example 4. Figure 3 displays λ^0 and the 0-retrospective ranking of our running example. In the prospective ranking (Figure 2), the nodes for state t on levels 1 and 2 are given rank 0, like other t-nodes. In the absence of a path forcing this, their retrospective rank is 2.

We are now ready to define a new construction, generating an NBW $\mathcal{A}_L$, which combines the benefits of the previous two constructions. The automaton $\mathcal{A}_L$ guesses the F-finite level k, and uses level rankings to check if the k-retrospective ranking is an odd ranking. We partition the operation of $\mathcal{A}_L$ into two stages. Until the level k, the NBW $\mathcal{A}_L$ is in the first stage, where it deterministically tracks preordered subsets. After level k, the NBW $\mathcal{A}_L$ moves to the second stage, where it tracks ranks. This stage is also deterministic. Consequently, the only nondeterminism in $\mathcal{A}_L$ is indeed the guess of k. Before defining $\mathcal{A}_L$, we need some definitions and notations.

Recall that $\mathbf{Q}$ denotes the set of preordered subsets of Q, and $\mathcal{R}_T^m$ the set of tight level rankings bounded by m. We distinguish between three types of transitions of $\mathcal{A}_L$: transitions within the first stage, transitions from the first stage to the second, and transitions within the second stage. The first type of transition is similar to the one taken in $\mathcal{A}_S$, by means of the σ-successor relation between preordered subsets. Below we explain in detail the other two types of transitions. Recall that in the retrospective ranking $\mathbf{r}^k$, each class in G' labeled $\top$ by λ^k is given a unique odd rank. Thus the rank of a node u depends on the number of $\top$-labeled classes larger than it, denoted $\alpha(u)$.

We begin with transitions where $\mathcal{A}_L$ moves between the stages: from a preordered subset $\langle S, \preceq \rangle$ to a level ranking. On level $k+1$, a node is labeled $\top$ iff it is an non-F-node. Thus for every $q \in S$, let $\beta(q) = |\{[v] \mid v \in S \setminus F, u \prec v\}|$ be the number of non-F-classes larger than q. We now define $\texttt{torank}\colon \mathbf{Q} \to \mathcal{R}_T^m$. Let $\texttt{torank}(\langle S, \preceq \rangle)$ be the tight level ranking f where for every q:

$$f(q) = \begin{cases} \bot & \text{if } q \notin S, \\ 2\beta(q) & \text{if } q \in S \cap F, \\ 2\beta(q)+1 & \text{if } q \in S \setminus F. \end{cases}$$

We now turn to transitions within the second stage, between level rankings. The rank of a node v is inherited from its predecessor u in G'. However, λ^k may label a finite class $\top$. If a $\top$-labeled class larger than u has no children, then $\alpha(u) \geq \alpha(v)$. In this case the rank of v decreases. Given a level ranking f, for every $q \in Q$ where $f(q) \neq \bot$, let $\gamma(q) = |\{f(q') \mid q' \in Q,\ f(q') \text{ is odd}, f(q') < f(q)\}|$ be the number of odd ranks in the range of f lower than $f(q)$. We define the function $\texttt{tighten}\colon \mathcal{R}^m \to \mathcal{R}_T^m$. Let $\texttt{tighten}(f)$ be the tight level ranking f' where for every q:

$$f'(q') = \begin{cases} \bot & \text{if } f(q) = \bot, \\ 2\gamma(q) & \text{if } f(q) \neq \bot \text{ and } q \in F, \\ 2\gamma(q)+1 & \text{if } f(q) \neq \bot \text{ and } q \notin F. \end{cases}$$

Note that if f is tight, then $f' = f$, and that while $\texttt{tighten}$ may merge two even ranks, it cannot merge two odd ranks.

For a level ranking f, letter $\sigma \in \Sigma$, and $q' \in Q$, let $\texttt{pred}(q', \sigma, f) = \{q \mid f(q) \neq \bot,\ q' \in \rho(q, \sigma)\}$ be the predecessors of q' given a non-$\bot$ rank by f. The lowest ranked element in this set corresponds to the predecessor in G with the maximal profile. With two exceptions, q' will inherit this lowest rank. First, $\texttt{tighten}$ might shift the rank down. Second, if q' is in F, it cannot be given an odd rank. For $n \in \mathbb{N}$, let $\lfloor n \rfloor_{even}$ be: n when n is even; and $n-1$ when n is odd. Define the *σ-successor of f* to be $\texttt{tighten}(f')$ where for every $q' \in Q$:

$$f'(q') = \begin{cases} \bot & \text{if } \texttt{pred}(q', \sigma, f) = \emptyset, \\ \lfloor \texttt{min}(\{f(q) \mid q \in \texttt{pred}(q', \sigma, f)\}) \rfloor_{even} & \text{if } \texttt{pred}(q', \sigma, f) \neq \emptyset \text{ and } q' \in F, \\ \texttt{min}(\{f(q) \mid q \in \texttt{pred}(q', \sigma, f)\}) & \text{if } \texttt{pred}(q', \sigma, f) \neq \emptyset \text{ and } q' \notin F. \end{cases}$$

▶ **Definition 16.** For an NBW $\mathcal{A} = \langle \Sigma, Q, Q^{in}, \rho, F \rangle$, let $\mathcal{A}_L$ be the NBW $\langle \Sigma, \mathbf{Q} \cup (\mathcal{R}_T^m \times 2^Q), Q_L^{in}, \rho_L, \mathcal{R}_T^m \times \{\emptyset\} \rangle$, where

- $Q_L^{in} = \{\langle Q^{in}, \preceq^{in} \rangle\}$ where $\preceq^{in}$ is such that for all $q, r \in Q^{in}$, $q \preceq r$ iff $q \notin F$ or $r \in F$.
- $\rho_L(\mathcal{S}, \sigma) = \{\mathcal{S}'\} \cup \{\langle \texttt{torank}(\mathcal{S}'), \emptyset \rangle\}$, where $\mathcal{S}'$ is the σ-successor of $\mathcal{S}$.
- $\rho_L(\langle f, O \rangle, \sigma) = \{\langle f', O' \rangle\}$ where f' is the σ-successor of f

$$\text{and } O' = \begin{cases} \rho(O, \sigma) \setminus \mathit{odd}(f') & \text{if } O \neq \emptyset, \\ \mathit{even}(f') & \text{if } O = \emptyset. \end{cases}$$

Theorem 17 follows from Lemmas 1 and 15 and Corollary 12.

▶ **Theorem 17.** *For every NBW $\mathcal{A}$, it holds that $L(\mathcal{A}_L) = \overline{L(\mathcal{A})}$.*

Analysis: Like the tight-ranking construction in Section 2, the automaton $\mathcal{A}_L$ operates in two stages. In both, the second stage is the set of tight level rankings and obligation sets. The tight-ranking construction uses sets of states in the first stage, and is bounded by the size of the second stage: $(0.96n)^n$ [5]. The automaton $\mathcal{A}_L$ replaces the first stage with preordered subsets. As the number of preordered subsets is $O((\frac{n}{e \ln 2})^n) \approx (0.53n)^n$ [21], the size of $\mathcal{A}_L$ remains bounded by $(0.96n)^n$. This can be improved to $(0.76n)^n$: see below. Further, $\mathcal{A}_L$ has a very restricted transition relation: states in the first stage only guess whether to remain in the first stage or move to the second, and have nondeterminism of degree 2. States in the second stage are deterministic. Thus the transition relation is linear in the number of states and size of the alphabet, and $\mathcal{A}_L$ is deterministic in the limit.

5 Discussion

We have unified the slice-based and rank-based approaches by phrasing the former in the language of run DAGs. This enables us to define and exploit a retrospective ranking, providing a deterministic-in-the-limit complementation construction that does not employ determinization. Experiments show that the more deterministic automata are, the better they perform in practice [18]. By avoiding determinization, we reduce the cost of such a construction from $(n^2/e)^n$ to $(0.76n)^n$ [14]. In addition, our transition generates a transition relation that is linear in the number of states and size of the alphabet. Schewe demonstrated how to achieve a similar linear bound on the transition relation, but the resulting relation is larger and is not deterministic in the limit [17].

The use of level rankings affords several improvements from existing research on the rank-based approach. First, the cut-point construction of Miyano and Hayashi [12] can be improved. Schewe's construction only checks one even rank at a time, reducing the size of the state space to $(0.76n)^n$, only an n^2 factor from the lower bound [17]. As Schewe's approach does not alter the progression of the level rankings, it could be applied directly to the second stage of Definition 16. The resulting construction inherits the asymptotic state-space complexity of [17]. Second, symbolically encoding a preorder is complicated. In contrast, ranks are easily encoded, and the transition between ranks is nearly trivial to implement in SMV [20]. By changing the states in first stage of $\mathcal{A}_L$ from preordered subsets to simple subsets, and guessing the appropriate transition to the second stage, we obtain a symbolic representation while maintaining determinism in the limit. This approach sacrifices the linear-sized transition relation, but this is less important in a symbolic encoding. Finally, the subsumption relations of Doyen and Raskin [4] could be applied to the second stage of the automaton, while it is unclear if it could be applied at all to the slice-based construction.

From a broader perspective, we find it very interesting that the prospective and retrospective approaches are so strongly related. Odd rankings seem to be inherently "prospective,"

depending on the descendants of nodes in the run DAG. By investigating the slice-based approach, we are able to pinpoint the dependency on the future to a single component: the F-free level. This suggests it may be possible to use odd rankings for determinization, automata with other accepting conditions, and automata on infinite trees.

References

1. C.S. Althoff, W. Thomas, and N. Wallmaier. Observations on determinization of Büchi automata. *TCS*, 363(2):224–233, 2006.
2. J.R. Büchi. On a decision method in restricted second order arithmetic. In *ICLMPS*, 1962.
3. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42:857–907, 1995.
4. L. Doyen and J.-F. Raskin. Antichains for the automata-based approach to model-checking. In *LMCS*, 5(1), 2009.
5. E. Friedgut, O. Kupferman, and M.Y. Vardi. Büchi complementation made tighter. In *FCS*, 17(4):851–867, 2006.
6. S. Gurumurthy, O. Kupferman, F. Somenzi, and M.Y. Vardi. On complementing nondeterministic Büchi automata. In *CHARME*, 96–110, 2003.
7. D. Kähler and Th. Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In *ICALP*, 724–735, 2008.
8. H. Karmarkar and S. Chakraborty. On minimal odd rankings for Büchi complementation. In *ATVA*, 228–243, 2009.
9. N. Klarlund. *Progress Measures and finite arguments for infinite computations*. PhD thesis, Cornell University, 1990.
10. O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. In *TOCL*, 2(2):408–429, 2001.
11. M. Michel. Complementation is more difficult with automata on infinite words. *CNET*, Paris, 1988.
12. S. Miyano and T. Hayashi. Alternating finite automata on ω-words. In *TCS*, 1984.
13. D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. In *TCS*, 141:69–107, 1995.
14. N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *LICS*, 255–264, 2006.
15. M.O. Rabin and D. Scott. Finite automata and their decision problems. In *IBM JRD*, 3:115–125, 1959.
16. S. Safra. On the complexity of ω-automata. In *FOCS*, 319–327, 1988.
17. S. Schewe. Büchi complementation made tight. In *STACS*, 661–672, 2009.
18. R. Sebastiani and S. Tonetta. "More deterministic" vs. "smaller" Büchi automata for efficient LTL model checking. In *CHARME*, 126–140, 2003.
19. A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. In *TCS*, 49:217–237, 1987.
20. D. Tabakov and M.Y. Vardi. Model checking Büchi specifications. In *LATA*, 2007.
21. M.Y. Vardi. Expected properties of set partitions. Research report, The Weizmann Institute of Science, 1980.
22. M.Y. Vardi. Automata-theoretic model checking revisited. In *VMCAI*, 137–150, 2007.
23. Q. Yan. Lower bounds for complementation of ω-automata via the full automata technique. In *ICALP*, 589–600, 2006.

Degrees of Lookahead in Context-free Infinite Games*

Wladimir Fridman, Christof Löding, and Martin Zimmermann

Lehrstuhl Informatik 7, RWTH Aachen University, Germany
{fridman,loeding,zimmermann}@automata.rwth-aachen.de

Abstract

We continue the investigation of delay games, infinite games in which one player may postpone her moves for some time to obtain a lookahead on her opponent's moves. We show that the problem of determining the winner of such a game is undecidable for deterministic context-free winning conditions. Furthermore, we show that the necessary lookahead to win a deterministic context-free delay game cannot be bounded by any elementary function. Both results hold already for restricted classes of deterministic context-free winning conditions.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs; F.4.3 Formal Languages

Keywords and phrases Infinite games, delay, context-free languages

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.264

1 Introduction

Many of today's problems in computer science are no longer concerned with programs that transform data and then terminate, but with non-terminating reactive systems which have to interact with an (possibly) antagonistic environment for an unbounded amount of time. The framework of infinite two-player games is a powerful and flexible tool to verify and synthesize such systems [6]. The seminal theorem of Büchi and Landweber [2] states that the winner of an infinite game on a finite arena with a regular winning condition can be determined and a corresponding finite-state winning strategy can be constructed effectively.

Ever since, this result was extended along different dimensions, e.g., the number of players, the type of arena, the type of winning condition, the type of interaction between the players (alternation or concurrency), zero-sum or nonzero-sum, and complete or incomplete information. In this work, we consider two of these dimensions, namely context-free winning conditions and the possibility for one player to delay her moves.

Walukiewicz showed that games with deterministic context-free winning conditions can be solved in exponential time [12]. On the other hand, the problem of determining the winner of a game with (non-deterministic) context-free winning condition is undecidable, which can be shown by a reduction from the universality problem for non-deterministic ω-pushdown automata (see, e.g. [5]).

In a delay game, one of the players can postpone her moves for some time, thereby obtaining a lookahead on the moves of her opponent. This allows her to win some games which she loses without lookahead, e.g., if her first move depends on the third move of her opponent. On the other hand, there are simple winning conditions that cannot be won with

* The third author was supported by the project *Games for Analysis and Synthesis of Interactive Computational Systems (GASICS)* of the *European Science Foundation*.

any finite lookahead, e.g., if her first move depends on all of the infinitely many moves of her opponent. Delay arises naturally when transmission of data in networks or components equipped with buffers are modeled. Also, from a theoretical point of view, uniformization of relations by continuous functions can be expressed and analyzed in this setting.

Hosch and Landweber proved that it is decidable, whether a game with regular winning condition can be won with bounded lookahead (i.e., only finitely many moves are postponed) [8]. This result was improved by Holtmann, Kaiser, and Thomas [7] who showed that if a player wins a game with arbitrary lookahead, then already with (doubly-exponential) bounded lookahead, and gave a streamlined decidability proof.

We consider games in which two players pick letters from alphabets Σ_I and Σ_O, respectively, thereby producing two infinite sequences α and β. Thus, a strategy for the second player induces a mapping $\sigma\colon \Sigma_I^\omega \to \Sigma_O^\omega$. It is winning for the second player if $(\alpha, \sigma(\alpha))$ is contained in the winning condition $L \subseteq \Sigma_I^\omega \times \Sigma_O^\omega$ for every α. In this case, we say that σ uniformizes L. In the classical setting, in which the players pick letters in alternation, the n-th letter of $\sigma(\alpha)$ depends only on the first n letters of α. A strategy with bounded lookahead induces a Lipschitz-continuous function σ (in the Cantor topology on Σ^ω) and a strategy with arbitrary lookahead induces a continuous function (or equivalently, a uniformly continuous function, as Σ^ω is compact).

Thus, stated in these terms, Hosch and Landweber proved the decidability of the uniformization problem for regular relations by Lipschitz-continuous functions. Holtmann, Kaiser, and Thomas proved the equivalence of the existence of a continuous uniformization function and the existence of a Lipschitz-continuous uniformization function for regular relations. They observe that this equivalence does not hold for deterministic context-free winning conditions by giving an example in which every other move has to be postponed, i.e., the lookahead grows linearly. They ask whether the winner of such a game can be determined effectively and what kind of lookahead is necessary to win. We answer these questions.

Firstly, by applying the result of Walukiewicz [12] it is easy to see that if we only allow a fixed bounded lookahead, then determining the winner is decidable. Then, we show that determining whether a given player wins the game with arbitrary lookahead is undecidable for deterministic context-free winning conditions. We complement this by giving a criterion to determine when this question is decidable, if the lookahead is restricted to some fixed class of functions. Intuitively, if there is no global bound on the lookahead provided by the class, then the problem of determining the winner is undecidable. If there is such a bound, then it follows from the decidability result that the winner can be determined effectively.

By closely inspecting the winning conditions constructed in the proofs, it follows that these undecidability results already hold for winning conditions given by visibly one-counter automata. These are pushdown automata that have a single stack symbol, i.e., their stack is essentially a counter which can be incremented, decremented, and tested for zero, and whose input letters control the behavior of the stack, i.e., a letter either always triggers a push transition, a pop transition, or a skip transition (the stack is not changed). Visibly pushdown automata are a popular choice to model recursive processes as their languages are closed under all boolean operations (as opposed to (deterministic) context-free languages in general), they can be determinized, and have good algorithmic properties (for a thorough discussion see [1], for the determinization procedure for visibly pushdown automata on ω-words see [9]). At the same time, we do not need the full strength of the parity or Muller acceptance condition: all winning conditions can be recognized by automata with weak acceptance conditions that refer only to the set of visited states, and not to the set of states visited infinitely often.

Finally, we consider the lookahead necessary to win delay games with deterministic context-free winning conditions. We present a deterministic context-free delay game which can be won if arbitrary lookahead is available, but not with lookahead that is bounded by an elementary function, i.e., bounded by a k-fold exponential for some fixed k. Again, the winning condition can be recognized by a visibly one-counter automaton with weak acceptance condition.

In terms of uniformization of relations by continuous functions, we show that it is undecidable to determine whether a deterministic context-free relation is uniformized by some continuous function and to determine whether it is uniformized by some Lipschitz-continuous function. Furthermore, the example by Holtmann, Kaiser, and Thomas shows that the equivalence between the existence of a continuous uniformization function and the existence of a Lipschitz-continuous uniformization function does not hold for deterministic context-free relations, as opposed to the regular case.

Our results show that, unlike in the regular case, adding lookahead to deterministic context-free games significantly changes their algorithmic properties. Bounded lookahead, which is sufficient to win regular games, can always be encoded into the winning condition, hence the classical algorithms to solve regular games without lookahead are still applicable. However, in case of deterministic context-free games, where unbounded lookahead is necessary, one cannot encode it into the winning condition while preserving its context-freeness. This is a reason why these games are hard to handle algorithmically.

This work is structured as follows: in Section 2, we introduce infinite games with delay formally and present the types of pushdown automata we consider. Then, in Section 3 we present the decidability and undecidability results. The lower bound for the lookahead is presented in Section 4. In Section 5, we conclude with some open questions.

2 Definitions

The set of non-negative integers is denoted by $\mathbb{N}$, and we define $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. For an integer $n > 0$ let $[n]$ denote the set $\{0, \dots, n-1\}$. We define the k-fold exponential function $\exp_k \colon \mathbb{N} \to \mathbb{N}$ inductively by $\exp_0(n) = n$, and $\exp_{k+1}(n) = 2^{\exp_k(n)}$. An alphabet Σ is a non-empty finite set of letters, Σ^* denotes the set of finite words over Σ, Σ^n denotes the set of words over Σ of length n, and Σ^ω denotes the set of infinite words over Σ. The empty word is denoted by ε. For $\alpha \in \Sigma^* \cup \Sigma^\omega$ and $n \in \mathbb{N}$ we write $\alpha(n)$ for the n-th letter of α. For $w \in \Sigma^*$ and $a \in \Sigma$, $|w|_a$ denotes the number of a's in w.

2.1 Games with Delay

Given a *delay function* $f \colon \mathbb{N} \to \mathbb{N}_+$ and an ω-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$, the game $\Gamma_f(L)$ is played by two players (the input player I and the output player O) in rounds $i = 0, 1, 2, \dots$ as follows: in round i, Player I picks a word $u_i \in \Sigma_I^{f(i)}$, then Player O picks one letter $v_i \in \Sigma_O$. We refer to the sequence $(u_0, v_0), (u_1, v_1), (u_2, v_2), \dots$ as a *play* of $\Gamma_f(L)$, which yields two infinite words $\alpha = u_0 u_1 u_2 \cdots$ and $\beta = v_0 v_1 v_2 \cdots$. Player O wins the play if and only if the *induced word* $\binom{\alpha(0)}{\beta(0)}\binom{\alpha(1)}{\beta(1)}\binom{\alpha(2)}{\beta(2)} \cdots$ is in L.

Given a delay function f, a *strategy* for Player I is a mapping $\tau_I \colon \Sigma_O^* \to \Sigma_I^*$ such that $|\tau_I(w)| = f(|w|)$, and a strategy for Player O is a mapping $\tau_O \colon \Sigma_I^* \to \Sigma_O$. Consider a play $(u_0, v_0), (u_1, v_1), (u_2, v_2), \dots$ of $\Gamma_f(L)$. Such a play is *consistent* with τ_I, if $u_i = \tau_I(v_0 \cdots v_{i-1})$ for every i; it is consistent with τ_O, if $v_i = \tau_O(u_0 \cdots u_i)$ for every i. A strategy τ for Player p is *winning* for her, if every play that is consistent with τ is won by Player p. In this case, we say Player p *wins* $\Gamma_f(L)$.

For a delay function $f\colon \mathbb{N} \to \mathbb{N}_+$ define its *distance function* d_f by $d_f(i) = \left(\sum_{j=0}^{i} f(j)\right) - (i+1)$, i.e., $d_f(i)$ is the lookahead attained by Player O after i rounds. We say that f is a *constant-delay function* with delay d, if $d_f(i) = d$ for all i; f is a *linear-delay function* with delay $k > 0$, if $d_f(i) = (i+1)(k-1)$ for all i; and we say that f is an *elementary-delay function*, if $d_f \in \mathcal{O}(\exp_k)$ for some fixed k. Here, "constant", "linear", and "elementary" refer to the lookahead for Player O, i.e., to the kind of distance function d_f, and not to the kind of delay function f.

▶ **Example 1.** Consider the language L over $\{0,1,\sharp\} \times \{0,1,\sharp\}$ containing the words of the form $\binom{0}{0}^{n_0}\binom{0}{1}^{n_0}\binom{\sharp}{\sharp}\binom{0}{0}^{n_1}\binom{0}{1}^{n_1}\binom{\sharp}{\sharp}\binom{0}{0}^{n_2}\binom{0}{1}^{n_2}\binom{\sharp}{\sharp}\cdots$ with $n_i > 0$ for all i, as well as all words whose first component is not of the form $0^{2n_0}\sharp 0^{2n_1}\sharp 0^{2n_2}\sharp\cdots$ with $n_i > 0$ for all i. In order to win a game with winning condition L, Player I has to produce infinitely many blocks of 0's, all of even length. If he does this, then Player O can still guarantee to win by answering the first half of every block by 0 and the second half by 1. To do so, she needs to know in which half of a block she currently is, which is guaranteed by the linear-delay function with delay 2.

2.2 Pushdown Automata

A *deterministic pushdown machine* (DPDM) is a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_{in}, \bot)$ where Q is a finite set of states, Σ is an input alphabet, Γ is a pushdown alphabet, $\bot \notin \Gamma$ is the initial pushdown symbol (let $\Gamma_\bot = \Gamma \cup \{\bot\}$), $q_{in} \in Q$ is the initial state, and $\delta\colon Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma_\bot \to Q \times \Gamma_\bot^*$ is a partial transition function satisfying for every $q \in Q$ and every $A \in \Gamma_\bot$: either $\delta(q,a,A)$ is defined for all $a \in \Sigma$ and $\delta(q,\varepsilon,A)$ is undefined, or $\delta(q,\varepsilon,A)$ is defined and $\delta(q,a,A)$ is undefined for all $a \in \Sigma$. We require that the initial pushdown symbol $\bot$ can neither be written on the stack nor be deleted from the stack.

A stack content is a word from $\Gamma^*\bot$, we assume the leftmost symbol to be the top of the stack. A *configuration* is a pair (q,γ) consisting of a state $q \in Q$ and a stack content $\gamma \in \Gamma^*\bot$. We write $(q, A\gamma) \overset{a}{\vdash} (q', \gamma'\gamma)$, if $(q',\gamma') = \delta(q,a,A)$ for $a \in \Sigma \cup \{\varepsilon\}$, $\gamma, \gamma' \in \Gamma_\bot^*$ and $A \in \Gamma_\bot$. For an ω-word $\alpha \in \Sigma^\omega$ an infinite sequence of configurations $\rho = (q_0,\gamma_0)(q_1,\gamma_1)(q_2,\gamma_2)\cdots$ is a *run* of $\mathcal{M}$ on α if and only if $(q_0,\gamma_0) = (q_{in},\bot)$ and for all $i \in \mathbb{N}$ exists $a_i \in \Sigma \cup \{\varepsilon\}$ such that $(q_i,\gamma_i) \overset{a_i}{\vdash} (q_{i+1},\gamma_{i+1})$ and $a_0a_1a_2\cdots = \alpha$. For a run ρ we define the set of states visited in ρ as $\mathrm{Occ}(\rho) = \{q \in Q \mid \exists j\colon \rho(j) = (q,\gamma_j) \text{ for some } \gamma_j\}$. Similarly, we define the set of states visited infinitely often in ρ as $\mathrm{Inf}(\rho) = \{q \in Q \mid \forall i \exists j > i\colon \rho(j) = (q,\gamma_j) \text{ for some } \gamma_j\}$.

A *parity pushdown automaton* (parity-DPDA) is a tuple $\mathcal{A} = (\mathcal{M}^\mathcal{A}, \mathrm{col})$ where $\mathcal{M}^\mathcal{A}$ is a DPDM and $\mathrm{col}\colon Q \to [d]$ is a priority function assigning to each state of $\mathcal{M}^\mathcal{A}$ a natural number. It accepts an ω-word $\alpha \in \Sigma^\omega$ if there exists a run ρ of $\mathcal{A}$ on α, such that $\min\{\mathrm{col}(q) \mid q \in \mathrm{Inf}(\rho)\}$ is even. The set of ω-words accepted by a parity-DPDA $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

2.2.1 Weak Automata

A *weak-parity pushdown automaton* (weak-parity-DPDA) is a tuple $\mathcal{A} = (\mathcal{M}^\mathcal{A}, \mathrm{col})$ where $\mathcal{M}^\mathcal{A}$ is a DPDM and $\mathrm{col}\colon Q \to [d]$ is a priority function assigning to each state of $\mathcal{M}^\mathcal{A}$ a natural number. It accepts an ω-word $\alpha \in \Sigma^\omega$ if there exists a run ρ of $\mathcal{A}$ on α, such that $\min\{\mathrm{col}(q) \mid q \in \mathrm{Occ}(\rho)\}$ is even. The set of ω-words accepted by a weak-parity-DPDA $\mathcal{A}$ is again denoted by $L(\mathcal{A})$.

A weak-parity-DPDA is an E-DPDA, if $\mathrm{col}(Q) \subseteq \{0,1\}$, i.e., a run is accepting if it visits a state with priority 0 at least once. A weak-parity-DPDA is an A-DPDA, if $\mathrm{col}(Q) \subseteq \{1,2\}$, i.e., a run is accepting if it never visits a state with priority 1.

2.2.2 Visibly and One-counter Automata

A DPDM is *one-counter*, if Γ is a singleton set. A *visibly pushdown alphabet* $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$ is an alphabet partitioned into three disjoint alphabets: Σ_c is a set of *calls*, Σ_r a set of *returns*, Σ_{int} is a set of *internal actions*. A *deterministic visibly pushdown machine* is a DPDM $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_{in}, \bot)$ where Σ is a visibly pushdown alphabet and the transition function is composed of three functions $\delta = \delta_c \cup \delta_r \cup \delta_{int}$ where $\delta_c \colon Q \times \Sigma_c \times \Gamma_\bot \to Q \times \Gamma$, $\delta_r \colon Q \times \Sigma_r \times \Gamma_\bot \to Q$, and $\delta_{int} \colon Q \times \Sigma_{int} \times \Gamma_\bot \to Q$. A deterministic visibly pushdown machine can be seen as a DPDM with transition function δ' by defining

- $\delta'(q, a, A) = (q', A'A)$, if $a \in \Sigma_c$ and $\delta_c(q, a, A) = (q', A')$,
- $\delta'(q, a, A) = \begin{cases} (q', \varepsilon) & \text{if } A \neq \bot, \\ (q', \bot) & \text{if } A = \bot, \end{cases}$ if $a \in \Sigma_r$ and $\delta_r(q, a, A) = q'$, and
- $\delta'(q, a, A) = (q', A)$, if $a \in \Sigma_{int}$ and $\delta_{int}(q, a, A) = q'$.

Note that a deterministic visibly pushdown machine treats a return symbol as an internal action when the stack is empty.

Any type of DPDA introduced above is called *visibly one-counter*, denoted by DV1CA, if the underlying DPDM is visibly and one-counter. We prefix the abbreviation DV1CA by the type of acceptance condition of the automaton (parity, weak-parity, E, or A).

3 Decision Problems

In this section, we consider various decision problems regarding delay games with context-free winning conditions. We begin by showing that the winner for a fixed delay function with bounded distance function can be determined effectively. Then, we show that determining whether Player O has a winning strategy for some finite delay is undecidable. As a corollary we obtain that determining whether Player O has a winning strategy for some constant-delay or linear-delay function is undecidable, too. We conclude by giving a general criterion to classify the sets $\mathcal{F}$ of delay functions for which it is decidable whether Player O can win a given delay game with some function from $\mathcal{F}$.

As we consider winning conditions L that are recognizable by parity-DPDA, $\Gamma_f(L)$ can be modeled as a parity game on a countable arena with finitely many priorities. Since parity games are determined [4, 10], we conclude that delay games are also determined.

▶ **Theorem 2.** *Let $\mathcal{A}$ be a parity-DPDA and $f \colon \mathbb{N} \to \mathbb{N}_+$. Then, $\Gamma_f(L(\mathcal{A}))$ is determined.*

By encoding the delay into the winning condition the following theorem is obtained. Note that the property "$\{i \mid f(i) \neq 1\}$ is finite" covers all constant-delay functions f.

▶ **Theorem 3.** *The following problem is decidable:*
Input: *Parity-DPDA $\mathcal{A}$ and $f \colon \mathbb{N} \to \mathbb{N}_+$ such that $\{i \mid f(i) \neq 1\}$ is finite.*
Question: *Does Player O win $\Gamma_f(L(\mathcal{A}))$?*

Proof. Let $\sharp$ be a letter not occurring in Σ_O. For a word $\beta = \beta(0)\beta(1)\beta(2)\cdots \in \Sigma_O^\omega$, define $\mathrm{shift}_f(\beta) = \beta'(0)\beta'(1)\beta'(2)\cdots \in (\Sigma_O \cup \{\sharp\})^\omega$ by

$$\beta'(n) = \begin{cases} \beta(i) & \text{if } n = \left(\sum_{j=0}^{i} f(j)\right) - 1 \text{ for some } i, \\ \sharp & \text{otherwise.} \end{cases}$$

Figure 1 shows an example for this operation.

β:	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	$\cdots$
$\text{shift}_f(\beta)$:	$\sharp$	$\sharp$	a_0	a_1	$\sharp$	a_2	$\sharp$	a_3	a_4	a_5	$\cdots$

Figure 1 The shift_f-operation where $f(0) = 3$, $f(2) = 2$, $f(3) = 2$, and $f(i) = 1$ otherwise.

Now, let $L = L(\mathcal{A})$ and define

$$L_{\text{shift}_f} = \left\{ \binom{\alpha(0)}{\beta'(0)}\binom{\alpha(1)}{\beta'(1)}\binom{\alpha(2)}{\beta'(2)}\cdots \;\middle|\; \binom{\alpha(0)}{\beta(0)}\binom{\alpha(1)}{\beta(1)}\binom{\alpha(2)}{\beta(2)}\cdots \in L \text{ and } \text{shift}_f(\beta(0)\beta(1)\beta(2)\cdots) = \beta'(0)\beta'(1)\beta'(2)\cdots \right\}.$$

Since L is deterministic context-free and $\{i \mid f(i) \neq 1\}$ is finite, L_{shift} is also deterministic context-free. Furthermore, Player p wins $\Gamma_f(L)$ if and only if she wins the game $\Gamma_g(L_{\text{shift}_f})$ where $g(i) = 1$ for all $i \in \mathbb{N}$. As $\Gamma_g(L_{\text{shift}_f})$ is a game without delay, its winner can be determined effectively [12]. ◀

It turns out that Theorem 3 is the most general decidability result for arbitrary parity-DPDA: relaxing the finiteness condition on $\{i \mid f(i) \neq 1\}$ makes the problem undecidable (see Theorem 6).

We continue with the undecidability results which are obtained by a reduction from the halting problem for 2-register machines. A 2-register machine $\mathcal{R}$ is a list $(0\colon I_0), \ldots, (k-2\colon I_{k-2}), (k-1\colon \texttt{STOP})$, where the first entry of a pair $(\ell\colon I_\ell)$ is the line number and the second one is the instruction, which is of the form `INC(X`$_i$`)`, `DEC(X`$_i$`)`, or `IF X`$_i$`=0 GOTO m` where $i \in \{0,1\}$ is the number of a register and $m \in [k]$. A configuration of $\mathcal{R}$ is a tuple (ℓ, n_0, n_1) where $\ell \in [k]$ is a line number and $n_0, n_1 \in \mathbb{N}$ are the contents of the registers. The semantics are defined in the obvious way with the convention that a decrease of a register holding a zero has no effect. We say that $\mathcal{R}$ halts, if it reaches a configuration $(k-1, n_0, n_1)$ for some $n_0, n_1 \in \mathbb{N}$ when started with the initial configuration $(0,0,0)$. It is well-known that the halting problem for 2-register machines is undecidable [11].

▶ **Theorem 4.** *The following problem is undecidable:*
Input: *Parity-DPDA $\mathcal{A}$.*
Question: *Is there a delay function f such that Player O wins $\Gamma_f(L(\mathcal{A}))$?*

Proof. We proceed by a reduction from the halting problem for 2-register machines. Given such a machine $\mathcal{R} = (0\colon I_0), \ldots, (k-2\colon I_{k-2}), (k-1\colon \texttt{STOP})$, we encode a configuration (ℓ, n_0, n_1) by the word $\ell r_0^{n_0} r_1^{n_1}$. Note that if c encodes a configuration, then we have $|c'| \leq |c| + 1$ for the encoding c' of the successor configuration.

Now, define $\text{Conf} = \{\ell r_0^{n_0} r_1^{n_1} \mid \ell \in [k],\ n_0, n_1 \in \mathbb{N}\}$, $\text{Conf}_0 = 0$, and consider the following game specification over $\Sigma_I \times \Sigma_O$, where $\Sigma_I = \{\sharp, r_0, r_1\} \cup [k]$ and $\Sigma_O = \{N, E_0, E_1, L\}$: Player I builds up a word of the form $\sharp\text{Conf}_0(\sharp\text{Conf})^\omega$ (if he does not, he loses). Consider such a word $\sharp c_0 \sharp c_1 \sharp c_2 \sharp \cdots$ with $c_0 = \text{Conf}_0$ and $c_i \in \text{Conf}$ for all $i > 0$. In order to win, Player O has to find a pair c_j, c_{j+1} such that c_{j+1} does not encode the successor configuration of the configuration encoded by c_j. To do this, she indicates at each position where Player I has played a $\sharp$ whether she believes that the following two configurations are indeed successive configurations (by playing the letter N) or whether she claims an error (by playing E_0, E_1, L indicating that the first register, the second register, or the line number is not updated correctly). At any other position, she may pick an arbitrary letter.

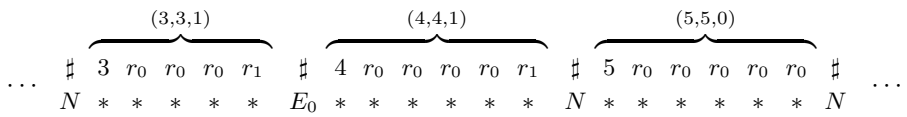


Figure 2 Part of a play encoding three configurations.

Figure 2 depicts the encoding of three configurations (here, $*$ denotes an arbitrary letter). Assuming that line 3 contains `INC(X0)` and line 4 contains `DEC(X1)`, then the first update is correct, while the second one is not: the first register is increased incorrectly, an error which is claimed by the letter E_0 in front of the second encoding.

This winning condition can be recognized by a parity-DPDA $\mathcal{A}_\mathcal{R}$. The automaton checks whether the first component is a word in $\sharp\mathrm{Conf}_0(\sharp\mathrm{Conf})^\omega$. If it encounters a letter $\binom{\sharp}{E_0}$, $\binom{\sharp}{E_1}$, or $\binom{\sharp}{L}$ it has to check the next two encodings $\ell r_0^{n_0} r_1^{n_1} \sharp \ell' r_0^{n_0'} r_1^{n_1'} \sharp$.

- Case $\binom{\sharp}{E_i}$ for $i \in \{0,1\}$: $\mathcal{A}_\mathcal{R}$ has to verify $n_i + s \neq n_i'$, where $s = 1$, if $I_\ell =$ `INC(Xi)`, $s = -1$, if $I_\ell =$ `DEC(Xi)` and $n_i > 0$, and $s = 0$ otherwise.
- Case $\binom{\sharp}{L}$: $\mathcal{A}_\mathcal{R}$ has to verify $\ell + 1 \neq \ell'$, if $I_\ell =$ `INC(Xi)`, $I_\ell =$ `DEC(Xi)`, or $I_\ell =$ `IF Xi=0 GOTO m` and $n_i > 0$; and $\mathcal{A}_\mathcal{R}$ has to verify $\ell' \neq m$, if $I_\ell =$ `IF Xi=0 GOTO m` and $n_i = 0$.

All these tests can be implemented in terms of a parity-DPDA $\mathcal{A}_\mathcal{R}$ that accepts a word if and only if the first component is not a word in $\sharp\mathrm{Conf}_0(\sharp\mathrm{Conf})^\omega$ or if the first occurrence of a letter E_0, E_1, or L in the second component correctly claims an error.

We show that $\mathcal{R}$ halts if and only if there exists a delay function f such that Player O wins the game $\Gamma_f(L(\mathcal{A}_\mathcal{R}))$.

Suppose $\mathcal{R}$ halts and consider the linear-delay function $f(i) = 6$ for all i. We claim that Player O has a winning strategy for $\Gamma_f(L(\mathcal{A}_\mathcal{R}))$ which finds the first error introduced by Player I. In round 0 Player I chooses 6 letters which are sufficient for Player O to check whether Player I has encoded the initial configuration and its successor configuration, as the length of such an encoding is bounded by 6. Now consider a round $i > 0$: if the i-th input letter is not a $\sharp$, then Player O can choose an arbitrary output letter. So suppose that it is a $\sharp$ and that Player O has not yet signaled an error up to this position: Player I has produced a word $\sharp x \sharp y$ of length $6(i+1)$ where $|x| = i - 1$ and hence, $|y| = 5(i+1)$. Note that both x and y might contain the letter $\sharp$. Let c denote the last encoding of a configuration in x and c' the first encoding of a configuration in y. As Player O has not signaled an error at the previous $\sharp$, we know that c' is well-defined and that it is the encoding of the successor configuration of the configuration encoded by c. We have $|c| \leq |x| = i - 1$ and hence $|c'| \leq i$. Thus, the successor configuration of c' is encoded by at most $i + 1$ letters. As $i + (i+1) + 2 < 5(i+1)$ for all $i > 0$, in every round Player O has enough information to detect an error if one is introduced. This strategy is indeed winning for Player O as an error will eventually be introduced, since a halting configuration has no successor.

Now suppose $\mathcal{R}$ does not halt. Player I has a winning strategy in $\Gamma_f(L(\mathcal{A}_\mathcal{R}))$ for any function f by building up the word $\sharp c_0 \sharp c_1 \sharp c_2 \sharp \cdots$ where $c_0, c_1, c_2, \ldots$ are the encodings of the infinite run of $\mathcal{R}$ starting with the initial configuration. Hence, due to determinacy, Player O does not win $\Gamma_f(L(\mathcal{A}_\mathcal{R}))$. ◀

The game induced by $L(\mathcal{A}_\mathcal{R})$ can be won by Player O with some suitable constant-delay function or with the linear-delay function with delay 6 if and only if $\mathcal{R}$ halts. Hence, we obtain the following corollary.

▶ **Corollary 5.** *The following problems are undecidable:*

1. ***Input:*** *Parity-DPDA $\mathcal{A}$.*
   Question: *Is there a constant-delay function f such that Player O wins $\Gamma_f(L(\mathcal{A}))$?*
2. ***Input:*** *Parity-DPDA $\mathcal{A}$.*
   Question: *Is there a linear-delay function f such that Player O wins $\Gamma_f(L(\mathcal{A}))$?*
3. ***Input:*** *Parity-DPDA $\mathcal{A}$ and $k \in \mathbb{N}$.*
   Question: *Let $f(i) = k$ for all i. Does Player O win $\Gamma_f(L(\mathcal{A}))$?*

By slightly modifying the game described above, one shows that all undecidability results hold even for winning conditions given by E-DV1CA. To this end, we supply Player O with additional letters C, R, Int and define $\Sigma_c = \Sigma_I \times \{C\}$, $\Sigma_r = \Sigma_I \times \{R\}$, and $\Sigma_{int} = \Sigma_I \times \{Int, N, L, E_0, E_1\}$, i.e., Player O controls the behavior of the stack. As soon as she has answered a $\sharp$ by one of the letters E_0, E_1, L she has to use the letters C, R, Int to enable the automaton to compare the respective parts of the following two encodings. If she fails to do so, she loses immediately. This can be implemented by means of a finite automaton, which can be combined with the parity-DPDA $\mathcal{A}_{\mathcal{R}}$ to obtain a visibly parity-DPDA. Furthermore, all necessary tests can be implemented using a single stack symbol. Finally, by modifying the game specification such that the first component is no longer required to be of the form $\sharp\text{Conf}_0(\sharp\text{Conf})^\omega$, it can be recognized by an E-DV1CA. If Player I does not produce an infinite sequence of encodings of configurations, Player O has the possibility to claim an error and thereby win.

Based on the proofs of the previous theorems, we conclude this section by giving a general criterion to determine for a set $\mathcal{F}$ of delay functions whether it is decidable whether Player O wins a given delay game with some delay function from $\mathcal{F}$. We say that a set $\mathcal{F}$ of delay functions $f\colon \mathbb{N} \to \mathbb{N}_+$ is *bounded*, if there exists a $d \in \mathbb{N}$ such that for every $f \in \mathcal{F}$ and every $i \in \mathbb{N}$ we have $d_f(i) \leq d$, i.e., there is a global bound on the lookahead for Player O given by the functions in $\mathcal{F}$. Notice that since $\mathcal{F}$ is not part of the input, we can state the following theorem for any set of delay functions without having to represent the set effectively.

▶ **Theorem 6.** *Let $\mathcal{F}$ be a set of delay functions. The following problem is decidable if and only if $\mathcal{F}$ is bounded:*

Input: *Parity-DPDA $\mathcal{A}$.*
Question: *Does there exist an $f \in \mathcal{F}$ such that Player O wins $\Gamma_f(L(\mathcal{A}))$?*

Proof. Consider a bounded set $\mathcal{F}$ of delay functions. We define a partial order on delay functions as follows: $f \leq g$ if and only if $d_f(i) \leq d_g(i)$ for all i, i.e., g allows at any round at least as much lookahead as f does. Applying Dickson's Lemma [3] and the boundedness of $\mathcal{F}$ one shows that there exists a finite set of maximal elements $\mathcal{F}_{max} \subseteq \mathcal{F}$ (a function $f \in \mathcal{F}$ is maximal if for all $g \in \mathcal{F}$, $f \leq g$ implies $f = g$). We claim that there exists an $f \in \mathcal{F}$ such that Player O wins $\Gamma_f(L(\mathcal{A}))$ if and only if there exists an $g \in \mathcal{F}_{max}$ such that Player O wins $\Gamma_g(L(\mathcal{A}))$. As $\mathcal{F}_{max}$ is finite and every $g \in \mathcal{F}_{max}$ satisfies "$\{i \mid g(i) \neq 1\}$ is finite", the latter property can be decided by Theorem 3.

The implication from right to left is trivially true, so assume there exists an $f \in \mathcal{F} \setminus \mathcal{F}_{max}$ such that Player O wins $\Gamma_f(L(\mathcal{A}))$. Then, there is a function $g \in \mathcal{F}_{max}$ such that $f \leq g$, i.e., the function g admits Player O at least as much lookahead as f. Hence, a winning strategy for Player O in $\Gamma_f(L(\mathcal{A}))$ can easily be turned into a winning strategy for her in $\Gamma_g(L(\mathcal{A}))$.

Now consider an unbounded set $\mathcal{F}$ of delay functions, i.e., for every $d \in \mathbb{N}$ there exists an $f \in \mathcal{F}$ and an $i \in \mathbb{N}$ such that $d_f(i) > d$. We adapt the specification described in the proof of Theorem 4 by allowing Player O to postpone the beginning of the simulation of a

computation of $\mathcal{R}$ until she has attained enough lookahead to inspect the complete halting computation of $\mathcal{R}$ (if there exists one) before she has to indicate potential errors.

Given a 2-register machine $\mathcal{R}$ with k instructions, define Conf_0 and Conf as in the proof of Theorem 4, and consider the following game specification over $\Sigma_I \times \Sigma_O$, where $\Sigma_I = \{\sharp, r_0, r_1, \$\} \cup [k]$ and $\Sigma_O = \{N, E_0, E_1, L, S, \$\}$: Player I builds a word of the form $\$^*\mathrm{Conf}_0(\sharp\mathrm{Conf})^\omega$ or $\$^\omega$. If he does not adhere to the format, he loses. Player I may produce the word $\$^\omega$ if and only if Player O never plays the letter S to start the simulation. If Player O plays the letter S, then Player I has to play a word of the form $\$^*c_0\sharp c_1\sharp c_2\sharp\cdots$ with $c_0 = \mathrm{Conf}_0$ and $c_i \in \mathrm{Conf}$ for every $i > 0$. Again, in order to win, Player O has to find a pair c_j, c_{j+1} such that c_{j+1} does not encode the successor configuration of the configuration encoded by c_j. The mechanism to do so is similar to the one described in the proof of Theorem 4. We denote the parity-DPDA recognizing this winning condition by $\mathcal{A}'_{\mathcal{R}}$.

Suppose $\mathcal{R}$ halts after n computation steps. Then, the full computation of $\mathcal{R}$ is encoded by at most $d = \sum_{j=1}^{n+1}(j+1)$ letters. Let $f \in \mathcal{F}$ and $i \in \mathbb{N}$ such that $d_f(i) \geq d$. Player O has a winning strategy in $\Gamma_f(L(\mathcal{A}'_{\mathcal{R}}))$. In the first i rounds, she chooses \$. If Player I has picked in a round $j \leq i+1$ a word $u_j \neq \$^{f(j)}$, then Player O wins by playing \$ ad infinitum. Otherwise, she plays S in round $i+1$. Hence, in order to win Player I has to start simulating $\mathcal{R}$, say at position $j > i$. As d_f is non-decreasing, Player O has at least d letters lookahead when picking her letter in any round $j' \geq j$. As the machine halts, this lookahead enables her to detect an error which Player I has to introduce, since a halting configuration does not have a successor configuration.

If $\mathcal{R}$ does not halt, then Player I has a winning strategy in $\Gamma_f(L(\mathcal{A}'_{\mathcal{R}}))$ for every delay function $f \in \mathcal{F}$: as long as Player O has not played S, pick $\$^{f(i)}$ in round i. As soon as she has played S, he starts producing the word $\sharp c_0\sharp c_1\sharp c_2\sharp\cdots$, where $c_0, c_1, c_2, \ldots$ are encodings of the infinite run of $\mathcal{R}$ starting in the initial configuration. ◀

Using the ideas presented above, one can show that Theorem 6 holds even for weak-parity-DV1CA. However, E-acceptance and A-acceptance are not sufficient, since Player O has to be forced to play an S and Player I has to be forced to start the simulation after Player O played an S.

4 Lower Bounds on Delays

In this section we show that there exists a deterministic context-free winning condition L such that Player O wins the game $\Gamma_f(L)$ for some delay function f, but not for any elementary-delay function. To this end, we adapt the idea of the previous section: Player I produces an ω-word which can be decomposed into blocks on which a successor relation is defined. In order to win, Player O has to find a pair of consecutive blocks that are not in the successor relation and the game specification forces Player I to produce such an error at some point. In contrast to the specifications of the previous section, Player O does not signal a potential error in front of the i-th block, but with her i-th bit. By ensuring that a valid successor block is exponentially longer than its predecessor we obtain our result.

▶ **Theorem 7.** *There exists a parity-DPDA $\mathcal{A}$ and a delay function f such that Player O wins $\Gamma_f(L(\mathcal{A}))$, but Player I wins $\Gamma_{f'}(L(\mathcal{A}))$ for every elementary-delay function f'.*

Proof. Let $S_\sharp = \{\sharp_N, \sharp_D, \sharp_C\}$ and $S_\flat = \{\flat_N, \flat_H\}$ be two sets of signals for Player I and define $B = S_\flat 0(S_\flat 0^+)^*$ and $B_0 = S_\flat 0$. We say that a *block* $w = \flat_0 0 \flat_1 0^{n_1} \flat_2 \cdots \flat_{k-1} 0^{n_{k-1}} \in B$ has k $\flat$*-blocks*. Consider a word $\sharp_0 w_0 \sharp_1 w_1 \sharp_2 w_2 \sharp_3 \cdots$ where $w_0 \in B_0$ and $w_i \in B$, $\sharp_i \in S_\sharp$ for all i. We say that a block $w_i = \flat_0 0^{n_0} \flat_1 0^{n_1} \flat_2 \cdots \flat_{k-1} 0^{n_{k-1}}$ has a *doubling error* at position j in the

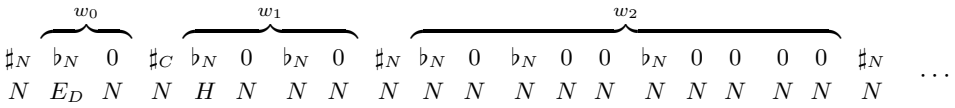


Figure 3 A play prefix with three blocks w_0, w_1, and w_2.

range $0 \leq j < k-1$ if $n_{j+1} \neq 2n_j$ (note that $n_0 = 1$ for every $w_i \in B$). The doubling error at position j in block w_i is *signaled*, if $\sharp_i = \sharp_D$, $\flat_j = \flat_H$, and $\flat_{j'} = \flat_N$ for all $j' < j$. We say that two consecutive blocks w_i and w_{i+1} constitute a *copy error*, if $|w_i| \neq |w_{i+1}|_{\flat_N} + |w_{i+1}|_{\flat_H}$ (i.e., in the absence of a copy error, w_{i+1} has $|w_i|$ $\flat$-blocks). For two blocks w_i and w_{i+1} the copy error is *signaled* if $\sharp_i = \sharp_C$.

Figure 3 depicts the encoding of three blocks in the first component. The blocks w_1 and w_2 constitute a copy error, as w_2 contains only three $\flat$-blocks, and not the required four (due to $|w_1| = 4$). This error is signaled by the letter $\sharp_C$ in front of w_1. Furthermore, the block w_1 contains a doubling error at position 0 which is not signaled.

Consider the following game specification over $\Sigma_I \times \Sigma_O$, where $\Sigma_I = \{0\} \cup S_\sharp \cup S_\flat$ and $\Sigma_O = \{N, E_D, E_C, H\}$: Player I builds a word $\alpha = \sharp_0 w_0 \sharp_1 w_1 \sharp_2 w_2 \sharp_3 \cdots \in S_\sharp B_0 (S_\sharp B)^\omega$ while Player O produces a word $\beta \in \Sigma_O^\omega$. Player O uses her letters to announce errors in α: if $\beta(i) = E_C$, then she claims that the pair w_i and w_{i+1} constitutes a copy error. If $\beta(i) = E_D$, then she claims that $w_i = \flat_0 0^{n_0} \flat_1 0^{n_1} \flat_2 \cdots \flat_{k-1} 0^{n_{k-1}}$ contains a doubling error at a position j, which she has to specify by answering $\flat_j$ by H (and every $\flat_{j'}$ for $j' < j$ not by H).

Going back to the example in Figure 3, we see that Player O claims the doubling error in w_1 by choosing $\beta(1) = E_D$ and identifying its position by playing H directly in front if it.

Player O wins a play if and only if the induced word $\binom{\alpha(0)}{\beta(0)}\binom{\alpha(1)}{\beta(1)}\binom{\alpha(2)}{\beta(2)}\cdots$ satisfies at least one of the following conditions.

- $\alpha \notin S_\sharp B_0 (S_\sharp B)^\omega$ (i.e., Player I does not adhere to the format).
- There exists an i such that $\beta(i) = E_D$, $\beta(j) = N$ for all $j < i$, $\sharp_j = \sharp_N$ for all $j < i$, the ℓ-th $\flat$ of w_i is answered by H (and ℓ is minimal with this property) and w_i contains a doubling error at position ℓ (i.e., Player O detects a doubling error in block w_i and there is no error signaled by Player I in front of any block w_j for some $j < i$. Note that the doubling error in block w_i may have been signaled by Player I).
- There exists an i such that $\beta(i) = E_C$, $\beta(j) = N$ for all $j < i$, $\sharp_j = \sharp_N$ for all $j < i$, and the pair w_i and w_{i+1} constitutes a copy error (i.e., Player O detects a copy error in blocks w_i and w_{i+1} and there is no error signaled by Player I in front of any block w_j for some $j < i$. Note that the copy error in the blocks w_i and w_{i+1} may have been signaled by Player I in front of w_i).
- There exists an i such that $\sharp_i = \sharp_D$ and $\sharp_j = \sharp_N$ for all $j < i$, and $\beta(j) = N$ for all $j \leq i$, and w_i does not contain $\flat_H$ or the two blocks following the first $\flat_H$ do not constitute a doubling error (i.e., Player I signals a doubling error but does not indicate its position correctly).
- There exists an i such that $\sharp_i = \sharp_C$ and $\sharp_j = \sharp_N$ for all $j < i$, and $\beta(j) = N$ for all $j \leq i$, and the pair w_i and w_{i+1} does not constitute a copy error (i.e., Player I signals a copy error without producing one).
- $\sharp_i = \sharp_N$ for all i (i.e., Player I never signals an error).

Hence, the play indicated in Figure 3 is winning for Player O as her correct claim $\beta(1) = E_D$ precedes the signal $\sharp_1 = \sharp_C$ of Player I.

Let $L = \{\rho \in (\Sigma_I \times \Sigma_O)^\omega \mid \rho \text{ is winning for Player } O\}$. We show that L can be recognized by a parity-DPDA $\mathcal{A}$: on an ω-word $\rho = \binom{\alpha(0)}{\beta(0)}\binom{\alpha(1)}{\beta(1)}\binom{\alpha(2)}{\beta(2)}\cdots$ where $\alpha = \sharp_0 w_0 \sharp_1 w_1 \sharp_2 w_2 \sharp_3 \cdots \in S_\sharp B_0 (S_\sharp B)^\omega$, $\mathcal{A}$ proceeds in four phases described below. If α is not of the required format or contains no letter $\sharp_C$ or $\sharp_D$, then ρ is accepted.

In the first phase, it prepares its stack to be able to find the beginning of w_i when starting at letter $\rho(i)$ as required in the second phase. To do so, it counts the number of letters processed so far minus the number of letters from $S_\sharp$ in the first component. This phase is stopped as soon as a letter $\sharp_C$ or $\sharp_D$ in the first component is read or a letter E_C or E_D in the second component is read. In the first case, the automaton jumps to phase four, in the second it starts with phase two.

The second phase starts if $\beta(i)$ is E_C or E_D for the first time. Then, the automaton uses the information on the stack to find the beginning of w_i by decreasing the stack every time a $\sharp_N$ in the first component is processed. If $\sharp_j = \sharp_C$ or $\sharp_j = \sharp_D$ for $j < i$ is processed, the automaton jumps to phase four. Otherwise, phase two continues until the beginning of w_i is reached. Then, $\mathcal{A}$ continues with phase three.

In phase three, $\mathcal{A}$ checks whether the error indicated by $\beta(i)$ occurs (for this purpose, it stores $\beta(i)$ at the beginning of phase two). If $\beta(i) = E_C$, then it checks whether w_i and w_{i+1} constitute a copy error. If $\beta(i) = E_D$, then it checks whether w_i contains a doubling error, which has to be indicated in the second component by an H right before the error. If the first H does not indicate an error correctly (or if none is read), then $\mathcal{A}$ rejects ρ. The automaton accepts in phase three if and only if the error indicated by $\beta(i)$ is detected.

Finally, in phase four $\mathcal{A}$ checks whether the error indicated by $\sharp_j$ occurs. If $\sharp_j = \sharp_C$, then it checks whether w_j and w_{j+1} constitute a copy error. If $\sharp_j = \sharp_D$, then it checks whether w_j contains a doubling error, which is signaled properly by a $\flat_H$ at the appropriate position. If the first $\flat_H$ does not indicate an error correctly (or if w_j does not contain a $\flat_H$), then $\mathcal{A}$ accepts ρ. The automaton accepts in phase four if and only if the error indicated by $\sharp_j$ is not detected.

All the tests described in phases three and four can be implemented by a parity-DPDA.

We continue by showing that there exists a delay function f such that Player O wins the game $\Gamma_f(L)$. To this end, note that the following holds true for two consecutive blocks w and w' not containing a copy or doubling error: $|w'| = 2^{|w|} + |w| - 1$. Hence, we define the auxiliary function g by $g(0) = 2$ and $g(n+1) = 2^{g(n)} + g(n) - 1$ for every $n \geq 0$. Now, define the delay function f by $f(0) = g(0) + g(1) + 3$ and $f(n) = g(n+1) + 1$ for every $n > 0$ (note that f is non-elementary). We claim that Player O has a winning strategy for $\Gamma_f(L)$: if Player I does not pick $\sharp_0 \flat_{00} 0 \sharp_1 \flat_{10} 0 \flat_{11} 00 \sharp_2$ in the first round, then he has committed some error within his first two blocks, which can be claimed by Player O with v_0. Now assume he has produced a play prefix $\sharp_0 w_0 \sharp_1 w_1 \sharp_2 \cdots \sharp_i w_i \sharp_{i+1}$ after round $i-1$ without introducing a doubling error in the blocks w_j for all $j < i$ and no copy error in the pairs w_j and w_{j+1} for all $j < i$. If he produces an x in the next round i that is of the form $w\sharp$ such that w_i and w do not constitute a copy error and if w_i does not contain a doubling error, then Player O picks $v_i = N$. Otherwise, she claims the error that occurs. This strategy is winning for Player O, as Player I is not able to signal and produce an error that cannot be claimed by Player O.

Finally, consider an elementary-delay function $f_e \in \mathcal{O}(\exp_k)$. Player I can always play blocks without introducing errors until the length of the block w_i exceeds the lookahead $\left(\sum_{j=0}^{i} f_e(j)\right) - i$ of Player O. At such a position, Player O has to make a claim concerning a block which is not completed yet. So, Player I signals a doubling error for this incomplete block. If Player O does not claim a doubling error, then he can introduce a doubling error while completing the block. Then, Player I wins, if he sticks to the input format, since he is

the first to claim an error. Vice versa, if Player O claims the doubling error, then Player I does not introduce a doubling error while completing the block. Then he continues to stick to the input format and wins, as his claim is preceded by the claim of Player O. ◀

Using ideas as presented in Section 3 one can show that Theorem 7 holds even for A-DV1CA. However, the game specification as described above, cannot be accepted by a visibly one-counter automaton: the problem arises if the automaton has to change from phase two to phase four. In this situation, the stack is not yet empty and the automaton has to check a claimed error. To do this using one stack symbol, the stack has to be emptied before the next letter is processed, which cannot be done by a visibly automaton, as it has no ε-transitions. To resolve this, we modify the game specification such that if this situation occurs (changing from phase two to phase four), Player O loses immediately. Player O has still the possibility to win by additionally never claiming an error in a block w_i if Player I already claimed an error in a block w_j for some $j < i$. This modified game specification is visibly one-counter. Finally, as a play is only winning for Player I, if he claims an existing error before Player O does, the set of winning plays for Player O can be accepted by an A-DV1CA.

5 Conclusion

In this paper we continued the investigation of delay games. We showed that determining the winner of deterministic context-free delay games is undecidable. Also, we presented a game that is won by Player O with finite delay, but the necessary lookahead is non-elementary. Both results already hold for the restricted class of winning conditions recognized by visibly one-counter automata with weak acceptance conditions.

Our undecidability results and lower bounds on the delay for visibly winning conditions hold even for the more restricted case where Player O controls the behavior of the stack (more formally, the membership of a letter to Σ_c, Σ_r, or Σ_{int} respectively, depends only on its second component). An interesting open question is whether these results also hold if Player I obtains control over the stack behavior, i.e., the first component of a letter determines to which alphabet it belongs. The following example shows that linear delay is necessary in this case, even for one-counter winning conditions.

Let $\Sigma_I = \{c, r\}$ and $\Sigma_O = \{0, 1\}$. We partition the alphabet $\Sigma_I \times \Sigma_O$ into the alphabets $\Sigma_c = \{c\} \times \Sigma_O$ and $\Sigma_r = \{r\} \times \Sigma_O$, i.e., Player I controls the behavior of the stack. Consider the following game specification L with $\rho = \binom{\alpha(0)}{\beta(0)}\binom{\alpha(1)}{\beta(1)}\binom{\alpha(2)}{\beta(2)} \cdots \in L$ if one of the following conditions holds:

- $\alpha(0) = r$, i.e., Player O wins immediately if Player I's first letter is a return,
- $|\alpha(0) \cdots \alpha(n)|_c > |\alpha(0) \cdots \alpha(n)|_r$ for all $n \in \mathbb{N}$, i.e., the stack is never empty, or
- for the minimal n such that $|\alpha(0) \cdots \alpha(n)|_c = |\alpha(0) \cdots \alpha(n)|_r$ we have $\beta(m) = 1$ and $\beta(m') = 0$ for all $m' < m$, where $m < n$ is the maximal position such that $\alpha(m) = c$, i.e., Player O indicates the last call position before the stack is empty for the first time.

This winning condition requires a linear-delay function with delay at least 2 for Player O, which is also sufficient for her to win. This is due to the fact that the stack height after processing n letters is bounded by n. Hence, Player I can play at most n returns before the stack is empty.

It is open whether linear delay is always sufficient for visibly winning conditions, if Player I controls the behavior of the stack. Moreover, it is open whether the winner of such a game can be determined effectively.

Acknowledgments

We want to thank Michael Holtmann for helpful discussions.

References

1 Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM, 56(3) (2009)
2 Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. Trans. Amer. Math. Soc. 138, 295–311 (1969)
3 Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. Amer. J. Math. 35(4), 413–422 (1913)
4 Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy (extended abstract). In: FOCS 91. IEEE (1991)
5 Finkel, O.: Topological properties of omega context-free languages. Theor. Comput. Sci., 262(1), 669–697 (2001)
6 Grädel, E., Thomas, W., Wilke, T. (eds): Automata, logics, and infinite Games. LNCS, vol. 2500, Springer, Heidelberg (2002)
7 Holtmann, M., Kaiser, Ł., Thomas, W.: Degrees of lookahead in regular infinite games. In: Ong, L., (ed), FOSSACS 2010. LNCS, vol. 6014, pp. 252–266, Springer, Heidelberg (2010)
8 Hosch, F., Landweber, L.H.: Finite delay solutions for sequential conditions. In: Nivat, M. (ed.), ICALP 1972, pp. 45–60. North-Holland, Amsterdam (1972)
9 Löding, C., Madhusudan, P., Serre, O.: Visibly pushdown games. In: Lodaya, K., Mahajan, M., (eds), FSTTCS 2004. LNCS, vol. 3328, pp. 408–420, Springer, Heidelberg (2004)
10 Mostowski, A.W.: Games with forbidden positions. Technical report 78, University of Gdańsk, Poland (1991)
11 Shepherdson, J. C., Sturgis, H. E.: Computability of recursive functions. J. ACM, 10(2), 217–255 (1963)
12 Walukiewicz, I.: Pushdown processes: games and model-checking. Inf. Comput., 164(2), 234–263 (2001)

L-Recursion and a new Logic for Logarithmic Space

Martin Grohe, Berit Grußien, André Hernich, and Bastian Laubner

Humboldt University Berlin
Germany
{grohe,grussien,hernich,laubner}@informatik.hu-berlin.de

Abstract

We extend first-order logic with counting by a new operator that allows it to formalise a limited form of recursion which can be evaluated in logarithmic space. The resulting logic LREC has a data complexity in LOGSPACE, and it defines LOGSPACE-complete problems like deterministic reachability and Boolean formula evaluation. We prove that LREC is strictly more expressive than deterministic transitive closure logic with counting and incomparable in expressive power with symmetric transitive closure logic STC and transitive closure logic (with or without counting). LREC is strictly contained in fixed-point logic with counting FP+C. We also study an extension $\mathsf{LREC}_=$ of LREC that has nicer closure properties and is more expressive than both LREC and STC, but is still contained in FP+C and has a data complexity in LOGSPACE.

Our main results are that LREC captures LOGSPACE on the class of directed trees and that $\mathsf{LREC}_=$ captures LOGSPACE on the class of interval graphs.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes; F.4.1 Mathematical Logic

Keywords and phrases Descriptive complexity, logarithmic space, fixed-point logics

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.277

1 Introduction

Descriptive complexity theory gives logical characterisations for most of the standard complexity classes. For example, Fagin's Theorem [6] states that a property of finite structures is decidable in NP if and only if it is definable in existential second-order logic Σ^1_1. More concisely, we say that Σ^1_1 *captures* NP. Similarly, Immerman [11] and Vardi [23] proved that fixed-point logic FP captures PTIME,[1] and Immerman [13] proved that deterministic transitive closure logic DTC captures LOGSPACE. However, these and all other known logical characterisations of PTIME and LOGSPACE and all other complexity classes below NP have a serious drawback — they only hold on ordered structures. (An *ordered structure* is a structure that has a distinguished binary relation which is a linear order of the elements of the structure.) The question of whether there are logical characterisations of these complexity classes on arbitrary, not necessarily ordered structures, is viewed as the most important open problem in descriptive complexity theory. For the class PTIME this problem goes back to Chandra and Harel's fundamental article [3] on query languages for relational databases.

[1] More precisely, Immerman and Vardi's theorem holds for *least fixed-point logic* and the equally expressive *inflationary fixed-point logic*. Our indeterminate FP refers to either of the two logics. For the counting extension FP+C considered below, it is most convenient to use an inflationary fixed-point operator. See any of the textbooks [4, 9, 14, 20] for details.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 277–291

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

For PTIME, at least partial positive results are known. The strongest of these say that fixed-point logic with counting FP+C captures PTIME on all classes of graphs with excluded minors [10] and on the class of interval graphs [17]. It is well-known that fixed-point logic FP (without counting) is too weak to capture PTIME on any natural class of structures that are not ordered. The idea that the extension FP+C by counting operators might remedy the weakness of FP goes back to Immerman [12]. Together with Lander he proved that FP+C captures PTIME on the class of trees [15]. Later, Cai, Fürer, and Immerman [2] proved that FP+C does not capture PTIME on all finite structures.

Much less is known for LOGSPACE. In view of the results described so far, an obvious idea is to try to capture LOGSPACE with the extension DTC+C of deterministic transitive closure logic DTC by counting operators. However, Etessami and Immerman [5] proved that (directed) tree isomorphism is not definable in DTC+C, not even in the stronger transitive closure logic with counting TC+C. Since Lindell [21] proved that tree isomorphism is decidable in LOGSPACE, this shows that DTC+C does not capture LOGSPACE.

We introduce a new logic LREC and prove that it captures LOGSPACE on directed trees. An extension $\mathsf{LREC}_=$ captures LOGSPACE on the class of interval graphs (and on the class of undirected trees). The logic LREC is an extension of first-order logic with counting by a "limited recursion operator". The logic is more complicated than the transitive closure and fixed-point logics commonly studied in descriptive complexity, and it may look rather artificial at first sight. To explain the motivation for this logic, recall that fixed-point logics may be viewed as extensions of first-order logic by fixed-point operators that allow it to formalise recursive definitions in the logics. LREC is based on an analysis of the amount of recursion allowed in logarithmic space computations. The idea of the limited recursion operator is to control the depth of the recursion by a "resource term", thereby making sure that we can evaluate the recursive definition in logarithmic space. Another way to arrive at the logic is based on an analysis of the classes of Boolean circuits that can be evaluated in LOGSPACE. We will take this route when we introduce the logic in Section 3.

LREC is easily seen to be (semantically) contained in FP+C. We show that LREC contains DTC+C, and as LREC captures LOGSPACE on directed trees, this containment is strict and, moreover, LREC is not contained in TC+C. Then we prove that undirected graph reachability is not definable in LREC. Hence LREC does not contain transitive closure logic TC, not even in its symmetric variant STC, and therefore LREC is strictly contained in FP+C.

It can be argued that our proof of the inability of LREC to express graph reachability reveals a weakness in our definition of the logic rather than a weakness of the limited recursion operator underlying the logic: LREC is not closed under (first-order) logical reductions. To remedy this weakness, we introduce an extension $\mathsf{LREC}_=$ of LREC. It turns out that undirected graph reachability is definable in $\mathsf{LREC}_=$ (this is a convenient side effect of the definition and not a deep result). Thus $\mathsf{LREC}_=$ strictly contains symmetric transitive closure logic with counting. We prove that $\mathsf{LREC}_=$ captures LOGSPACE on the class of interval graphs. To complete the picture, we prove that plain LREC, even if extended by a symmetric transitive closure operator, does not capture LOGSPACE on the class of interval graphs.

The paper is organised as follows: After giving the necessary preliminaries in Section 2, in Section 3 we introduce the logic LREC and prove that its data complexity is in LOGSPACE. Then in Section 4, we prove that directed tree isomorphism and canonisation are definable in LREC. As a consequence, LREC captures LOGSPACE on directed trees. In Section 5, we study the expressive power of LREC and prove that undirected graph reachability is not definable in LREC. The extension $\mathsf{LREC}_=$ is introduced in Section 6. Finally, our results on interval graphs are presented in Section 7. We close with concluding remarks and open problems. Due to space limitations, we defer many of the proofs to the full version of this paper.

2 Basic Definitions

$\mathbb{N}$ denotes the set of all non-negative integers. For all $m, n \in \mathbb{N}$, we define $[m, n] := \{p \in \mathbb{N} \mid m \leq p \leq n\}$, and $[n] := [1, n]$. For mappings $f\colon A \to B$, and tuples $\bar{a} = (a_1, \ldots, a_k)$ over A, we let $f(\bar{a}) := (f(a_1), \ldots, f(a_k))$. For a tuple $\bar{a} = (a_1, \ldots, a_k)$, we let $\tilde{a} := \{a_1, \ldots, a_k\}$.

A *vocabulary* is a finite set τ of relation symbols, where each $R \in \tau$ has a fixed arity $\mathrm{ar}(R)$. A *τ-structure* A consists of a non-empty finite set $V(A)$, its *universe*, and for each $R \in \tau$ a relation $R(A) \subseteq V(A)^{\mathrm{ar}(R)}$. For logics L, L' we write $\mathsf{L} \leq \mathsf{L}'$ if L is semantically contained in L', and $\mathsf{L} < \mathsf{L}'$ if this containment is strict.

All logics considered in this paper are extensions of *first-order logic with counting (*$\mathsf{FO{+}C}$*)*; see, e.g., [4, 9, 14, 20] for a detailed discussion of $\mathsf{FO{+}C}$ and its extensions. $\mathsf{FO{+}C}$ extends first-order logic by a counting operator that allows for counting the cardinality of $\mathsf{FO{+}C}$-definable relations. It lives in a two-sorted context, where structures A are equipped with a *number sort* $N(A) := [0, |V(A)|]$. $\mathsf{FO{+}C}$-variables are either *structure variables* that range over the universe of a structure, or *number variables* that range over the number sort. For each variable u, let $A^u := V(A)$ if u is a structure variable, and $A^u := N(A)$ if u is a number variable. Tuples $(u_1, \ldots, u_k)$ and $(v_1, \ldots, v_\ell)$ of variables are *compatible* if $k = \ell$, and for every $i \in [k]$ the variables u_i and v_i have the same type. Let $A^{(u_1,\ldots,u_k)} := A^{u_1} \times \cdots \times A^{u_k}$. An *assignment in A* is a mapping α from the set of variables to $V(A) \cup N(A)$, where for each variable u we have $\alpha(u) \in A^u$. For tuples $\bar{u} = (u_1, \ldots, u_k)$ of variables and $\bar{a} = (a_1, \ldots, a_k) \in A^{\bar{u}}$, the assignment $\alpha[\bar{a}/\bar{u}]$ maps u_i to a_i for each $i \in [k]$, and each variable $v \notin \tilde{u}$ to $\alpha(v)$.

$\mathsf{FO{+}C}$ is obtained by extending first-order logic with the following formula formation rules: $p \leq q$ is a formula for all number variables p, q; and $\#\bar{u}\,\psi = \bar{p}$ is a formula for all tuples $\bar{u}$ of variables, all tuples $\bar{p}$ of number variables, and all formulae ψ. Free variables are defined in the obvious way, with $\mathrm{free}(\#\bar{u}\,\psi = \bar{p}) := (\mathrm{free}(\psi) \setminus \tilde{u}) \cup \tilde{p}$. Formulas $\#\bar{u}\,\psi = \bar{p}$ hold in a structure A under an assignment α in A if $|\{\bar{a} \in A^{\bar{u}} \mid (A, \alpha[\bar{a}/\bar{u}]) \models \psi\}| = \langle \alpha(\bar{p}) \rangle_A$, where for tuples $\bar{n} = (n_1, \ldots, n_k) \in N(A)^k$ we let $\langle \bar{n} \rangle_A$ be the number

$$\langle \bar{n} \rangle_A := \sum_{i=1}^{k} n_i \cdot (|V(A)| + 1)^{i-1}.$$

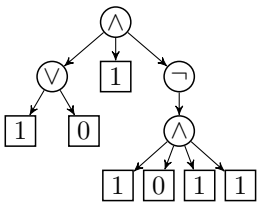
If A is understood from the context, we write $\langle \bar{n} \rangle$ instead of $\langle \bar{n} \rangle_A$.

We write $\varphi(u_1, \ldots, u_k)$ to denote a formula φ with $\mathrm{free}(\varphi) \subseteq \{u_1, \ldots, u_k\}$. Given a formula $\varphi(u_1, \ldots, u_k)$, a structure A and $a_1, \ldots, a_k \in A^{(u_1,\ldots,u_k)}$, we write $A \models \varphi[a_1, \ldots, a_k]$ if φ holds in A with u_i assigned to the element a_i, for each $i \in [k]$. We use similar notation for substitution: For a tuple $(v_1, \ldots, v_k)$ of variables that is compatible to $(u_1, \ldots, u_k)$, we let $\varphi(v_1, \ldots, v_k)$ be the result of substituting v_i for u_i for every $i \in [k]$. We write $\varphi[A, \alpha; \bar{u}]$ for the set of all tuples $\bar{a} \in A^{\bar{u}}$ with $(A, \alpha[\bar{a}/\bar{u}]) \models \varphi$.

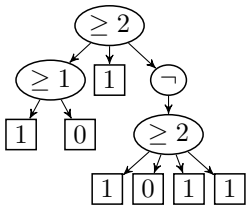
In many places throughout this paper we refer to various transitive closure and fixed-point logics (all mentioned in the introduction). Our results and remarks about the relation between these logics and our new logics LREC and $\mathsf{LREC}_=$ are relevant for a reader familiar with descriptive complexity theory to put our results in context, but they are not essential to follow the technical core of this paper. Therefore, we omit the definitions and refer the reader to the textbooks [4, 9, 14, 20].

3 The Logic LREC

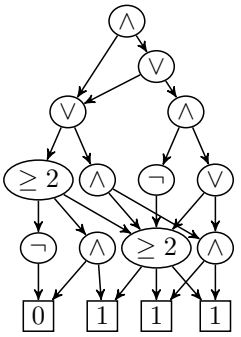
Let us start our development of LREC by looking at how certain kinds of Boolean circuits can be evaluated in $\mathsf{LOGSPACE}$.



The figure on the right shows a *Boolean formula*, i.e., a Boolean circuit whose underlying graph is a *tree*. It is easy to evaluate such circuits in LOGSPACE: Start at the output node, determine the value of the first child recursively, then determine the value of the second child, and so on. We only have to store the current node and its value (if it has been determined already), since the parent node and the next child of the parent (if any) are uniquely determined by the current node. It is known that Boolean formula evaluation is LOGSPACE-complete under NC^1-reductions [1].[2] In contrast, Boolean *circuit* evaluation is PTIME-complete.



Let us now turn to formulas with *threshold gates*, which may contain gates of the form "$\geq i$" for a number i in addition to the Boolean gates. An example is shown on the left. To evaluate such formulas in LOGSPACE, we again start at the root and evaluate the values of the children recursively. For each node we count how many 1-values we have seen already. To this end, when evaluating the values of the children of a node v, we begin with the child with the largest subtree and proceed to children with smaller subtrees. Note that the ith child of v in this order has a subtree of size at most s/i, where s is the size of the subtree of v. So, we can store a counter of up to $\log_2 i$ bits for the number of 1-values seen so far. It is easy to extend the algorithm to formulas with other *arithmetic gates* such as *modulo-gates*.



As a more complicated example, let us consider the following circuits. A circuit C has the *m-path property* if for all paths P in C the product of the in-degrees of the nodes on P is at most m. For example, formulas have the 1-path property, whereas the circuit on the right has the 16-path property. It is not hard to see that for every $k \geq 1$, circuits C having the $|C|^k$-path property can be evaluated in LOGSPACE. The $|C|^k$-path property here guarantees that in addition to a counter we can also store the path from the current node to the root, so that we can always find the parent of the current node. Another way of evaluating the circuit is to first "unravel" the circuit to a tree (i.e., a formula) which can be done in LOGSPACE due to the $|C|^k$-path property, and then to evaluate the formula as above.

The logic LREC allows it to recursively define sets X of tuples based on graphs G that have the $|G|^k$-path property for some $k \geq 1$.

We turn to the formal definition of the logic LREC. To define the syntax, let τ be a vocabulary. The set of all LREC[τ]-formulae is obtained by extending the formula formation rules of FO+C[τ] by the following rule: If $\bar{u}, \bar{v}, \bar{w}$ are compatible tuples of variables, $\bar{p}, \bar{r}$ are non-empty tuples of number variables, and φ_E and φ_C are LREC[τ]-formulae, then

$$\varphi := [\mathsf{lrec}_{\bar{u},\bar{v},\bar{p}}\ \varphi_E,\ \varphi_C](\bar{w}, \bar{r}) \tag{1}$$

is an LREC[τ]-formula, and we let $\mathrm{free}(\varphi) := (\mathrm{free}(\varphi_E) \setminus (\tilde{u} \cup \tilde{v})) \cup (\mathrm{free}(\varphi_C) \setminus (\tilde{u} \cup \tilde{p})) \cup \tilde{w} \cup \tilde{r}$.

To define the semantics of LREC[τ]-formulae, let A be a τ-structure and α an assignment in A. The semantics of LREC[τ]-formulae that are not of the form (1) is defined as usual.

Let φ be an LREC[τ]-formula of the form (1). We define a set $X \subseteq A^{\bar{u}} \times \mathbb{N}$ recursively as follows. We consider $E := \varphi_E[A, \alpha; \bar{u}, \bar{v}]$ as the edge relation of a directed graph with

[2] Here, the Boolean formula is represented by the list of all edges plus gate types in the circuit representing the formula.

vertex set $V := A^{\bar{u}}$. Moreover, for each vertex $\bar{a} \in V$ we think of the set $C(\bar{a}) := \{\langle \bar{n} \rangle \mid \bar{n} \in \varphi_C[A, \alpha[\bar{a}/\bar{u}]; \bar{p}]\}$ of integers as the label of $\bar{a}$. Let $\bar{a}E := \{\bar{b} \in V \mid \bar{a}\bar{b} \in E\}$ and $E\bar{b} := \{\bar{a} \in V \mid \bar{a}\bar{b} \in E\}$. Then, for all $\bar{a} \in V$ and $\ell \in \mathbb{N}$,

$$(\bar{a}, \ell) \in X \;:\Longleftrightarrow\; \ell > 0 \text{ and } \left| \left\{ \bar{b} \in \bar{a}E \;\middle|\; \left(\bar{b}, \left\lfloor \frac{\ell - 1}{|E\bar{b}|} \right\rfloor\right) \in X \right\} \right| \in C(\bar{a}).$$

Notice that X contains only elements $(\bar{a}, \ell)$ with $\ell > 0$. Hence, the recursion eventually stops at $\ell = 0$. We call X the *relation defined by φ in (A, α)*. Finally, we let

$$(A, \alpha) \models \varphi \;:\Longleftrightarrow\; \big(\alpha(\bar{w}), \langle \alpha(\bar{r}) \rangle\big) \in X.$$

▶ **Example 3.1** (Boolean circuit evaluation)**.** Let $\sigma := \{E, P_\wedge, P_\vee, P_\neg, P_0, P_1\}$. A Boolean circuit C may be viewed as a σ-structure, where $E(C)$ is the edge relation of C, and $P_\star(C)$ contains all $\star$-gates for $\star \in \{\wedge, \vee, \neg, 0, 1\}$. If C has the $|C|$-path-property, then $\exists r_1, r_2\, [\mathsf{lrec}_{x,y,p}\, E(x,y),\ \varphi_C](z, (r_1, r_2))$ with $\varphi_C(x,p) := (P_\wedge(x) \wedge \#y\, E(x,y) = p) \vee (P_\vee(x) \wedge$ "$p > 0$") $\vee$ ($P_\neg(x) \wedge$ "$p = 0$") $\vee\, P_1(x)$ states that gate z evaluates to 1. ◀

▶ **Example 3.2** (Deterministic transitive closure)**.** Let $\psi(\bar{u}, \bar{v})$ be an LREC$[\tau]$-formula, and let $\bar{s}, \bar{t}$ be tuples of variables such that $\bar{u}, \bar{v}, \bar{s}, \bar{t}$ are pairwise compatible. We give a formula φ such that for any τ-structure A and assignment α in A, we have $(A, \alpha) \models \varphi(s, t)$ iff in the graph $G = (V, E)$ defined by $V := A^{\bar{u}}$ and $E := \psi[A, \alpha; \bar{u}, \bar{v}]$ there is a *deterministic path* from $\alpha(\bar{s})$ to $\alpha(\bar{t})$, i.e., a path $v_1, \ldots, v_n$ from $\alpha(\bar{s})$ to $\alpha(\bar{t})$ such that for every $i \in [n-1]$, v_{i+1} is the unique out-neighbour of v_i. This is the same as reversing the edges of G and finding a path $v_n, \ldots, v_1$ from $\alpha(\bar{t})$ to $\alpha(\bar{s})$ such that for every $i \in [n-1]$, v_{i+1} is the unique in-neighbour of v_i. Therefore,

$$\varphi \;:=\; \exists \bar{r}\, [\mathsf{lrec}_{\bar{u},\bar{v},\bar{p}}\, \varphi_E(\bar{u}, \bar{v}),\ \varphi_C(\bar{u}, \bar{p})](\bar{t}, \bar{r}), \tag{2}$$

where $\bar{p}$ and $\bar{r}$ are $(|\bar{u}| + 1)$-tuples of number variables, and

$$\varphi_E(\bar{u}, \bar{v}) \;:=\; \psi(\bar{v}, \bar{u}) \wedge \forall \bar{u}'(\psi(\bar{v}, \bar{u}') \to \bar{u}' = \bar{u}), \qquad \varphi_C(\bar{u}, \bar{p}) \;:=\; \bar{u} = \bar{s} \vee (\bar{u} \neq \bar{s} \wedge \bar{p} \neq \bar{0}).$$

In the following, we use $[\mathsf{dtc}_{\bar{u},\bar{v}}\, \psi](\bar{s}, \bar{t})$ as an abbreviation for the LREC-formula in (2). ◀

The following theorem shows that the data complexity of LREC is in LOGSPACE.

▶ **Theorem 3.3.** *For every vocabulary τ, and every* LREC$[\tau]$*-formula φ there is a deterministic logspace Turing machine that, given a τ-structure A and an assignment α in A, decides whether $(A, \alpha) \models \varphi$.*

▶ Remark. It follows from Example 3.2 that DTC+C $\leq$ LREC. This containment is strict as directed tree isomorphism is definable in LREC (we will show this in the next section), but not in DTC+C. On the other hand, it is easy to see that the relation X defined by an LREC-formula of the form (1) in an interpretation (A, α) can be defined in fixed point logic with counting FP+C. Hence, LREC $\leq$ FP+C, and this containment is strict since we show in Section 5 that undirected graph reachability is not LREC-definable.

4 Capturing Logspace on Directed Trees

In this section we show that LREC captures LOGSPACE on the class of all directed trees. Our construction is based on Lindell's LOGSPACE tree canonisation algorithm [21]. Note, however, that Lindell's algorithm makes essential use of a linear order on the tree's vertices

that is given implicitly by the encoding of the tree. Here we do not have such a linear order, so we cannot directly translate Lindell's algorithm to an LREC-formula. We show that we can circumvent using the linear order if we have a formula for directed tree isomorphism. Hence, our first task is to construct such a formula.

4.1 Directed Tree Isomorphism

Let T be a directed tree. For every $v \in V(T)$ let T_v be the subtree of T rooted at v, let $\text{size}(v) := |V(T_v)|$ be the *size* of v, and let $\#_s(v)$ be the number of children of v of size s. We construct an $\mathsf{LREC}[\{E\}]$-formula $\varphi_{\cong}(x, y)$ that is true in a directed tree T with interpretations $v, w \in V(T)$ of x, y if and only if $T_v \cong T_w$. We assume that $|V(T)| \geq 4$, but it is easy to adapt the construction to directed trees with less than 4 vertices.

We implement the following recursive procedure to check whether $T_v \cong T_w$:

1. If $\text{size}(v) \neq \text{size}(w)$ or if $\#_s(v) \neq \#_s(w)$ for some $s \in [0, |V(T_v)| - 1]$, return "$T_v \ncong T_w$".
2. If for all children $\hat{v}$ of v there is a child $\hat{w}$ of w and a number k such that
   a. $T_{\hat{v}} \cong T_{\hat{w}}$,
   b. there are exactly k children $\mathring{w}$ of w with $T_{\hat{v}} \cong T_{\mathring{w}}$, and
   c. there are exactly k children $\mathring{v}$ of v with $T_{\mathring{v}} \cong T_{\hat{w}}$,

   then return "$T_v \cong T_w$".
3. Return "$T_v \ncong T_w$".

Clearly, this procedure outputs "$T_v \cong T_w$" if and only if $T_v \cong T_w$.

To simplify the presentation we fix a directed tree T and an assignment α in T, but the construction will be uniform in T and α.

We construct a directed graph $G = (V, E)$ with labels $C(v) \subseteq \mathbb{N}$ for each $v \in V$ as follows. Let $V := N(T) \times V(T)^4 \times N(T)$. The first component of each vertex is its *type*; the meaning of the other components will become clear soon. Although G will not be a tree, it is helpful to think of it as a *decision tree* for deciding $T_v \cong T_w$. For each pair $(v, w) \in V(T)^2$, we designate the vertex $\bar{a}_{v,w} = (0, v, w, v, w, 0)$ to stand for "$T_v \cong T_w$". Let us call (v, w) *easy* if v, w satisfy the condition in line 1 of the procedure (i.e., $\text{size}(v) \neq \text{size}(w)$, or $\#_s(v) \neq \#_s(w)$ for some $s \in [0, |V(T_v)| - 1]$). Note that the set of all such easy pairs is LREC-definable.[3] If (v, w) is easy, then $\bar{a}_{v,w}$ has no outgoing edges and $C(\bar{a}_{v,w}) = \emptyset$. On the other hand, if (v, w) is not easy, then G contains the following edges and labels (see Figure 1 for an illustration):

- The vertex $\bar{a}_{v,w}$ has an outgoing edge to $\bar{a}_{v,w,\hat{v}} := (1, v, w, \hat{v}, w, 0)$, for each child $\hat{v}$ of v. Furthermore, $C(\bar{a}_{v,w}) = \{\#\text{ of children of } v\}$. This corresponds to "for all children $\hat{v}$ of $v\ldots$" in the above procedure's step 2.
- The vertex $\bar{a}_{v,w,\hat{v}}$ has an outgoing edge to $\bar{a}_{v,w,\hat{v},\hat{w},k} := (2, v, w, \hat{v}, \hat{w}, k)$, for each child $\hat{w}$ of w with $\text{size}(\hat{w}) = \text{size}(\hat{v})$ and each $k \in [\#_{\text{size}(\hat{v})}(v)]$. Furthermore, $C(\bar{a}_{v,w,\hat{v}}) = N(T) \setminus \{0\}$. This branching corresponds to "$\ldots$there is a child $\hat{w}$ of w and a number k such that$\ldots$".
- The vertex $\bar{a}_{v,w,\hat{v},\hat{w},k}$ has an outgoing edge to $\bar{a}_{\hat{v},\hat{w}}$. If $\hat{v}$ is the only child of v of size $\text{size}(\hat{v})$, then this is the only outgoing edge, and we let $C(\bar{a}_{v,w,\hat{v},\hat{w},k}) = \{1\}$. Otherwise, there are additional outgoing edges to $\bar{a}^i_{v,w,\hat{v},\hat{w},k} = (3 + i, v, w, \hat{v}, \hat{w}, k)$ for $i \in \{0, 1\}$, and we let $C(\bar{a}_{v,w,\hat{v},\hat{w},k}) = \{3\}$. This corresponds to conditions 2a–2c.
- The vertex $\bar{a}^0_{v,w,\hat{v},\hat{w},k}$ has outgoing edges to $\bar{a}_{\hat{v},\mathring{w}}$ for each child $\mathring{w}$ of w of size $\text{size}(\hat{v})$, and $\bar{a}^1_{v,w,\hat{v},\hat{w},k}$ has outgoing edges to $\bar{a}_{\mathring{v},\hat{w}}$ for each child $\mathring{v}$ of v of size $\text{size}(\hat{w}) = \text{size}(\hat{v})$.

[3] Using the dtc-operator from Example 3.2 we can construct an $\mathsf{LREC}[\{E\}]$-formula defining the descendant relation between vertices in a directed tree, and using this formula it is easy to determine the size and the number of children of size s of a vertex.

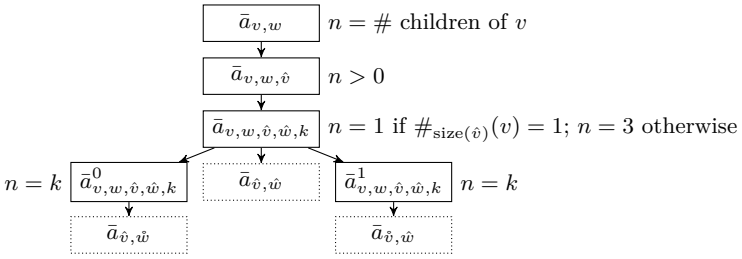


Figure 1 Sketch of "decision tree" for deciding $T_v \cong T_w$. Here, $\hat{v}, \mathring{v}$ range over the children of v; $\hat{w}, \mathring{w}$ range over the children of w; and $k \in [\#_{\mathrm{size}(\hat{v})}(v)]$. Moreover, $\hat{v}, \mathring{v}, \hat{w}, \mathring{w}$ all have the same size. Labels indicate which integers n belong to the set $C(\bar{a})$ labelling each vertex $\bar{a}$. If $\hat{v}$ is the only child of v of size $\mathrm{size}(\hat{v})$, then $\bar{a}_{\hat{v},\hat{w}}$ is the only child of $\bar{a}_{v,w,\hat{v},\hat{w},k}$.

Furthermore, $C(\bar{a}^i_{v,w,\hat{v},\hat{w},k}) = \{k\}$. The vertex $\bar{a}^i_{v,w,\hat{v},\hat{w},k}$ corresponds to condition 2b for $i = 0$, and to 2c for $i = 1$.

From the above description it should be easy to construct $\mathsf{LREC}[\{E\}]$-formulae $\varphi_E(\bar{u}, \bar{u}')$ and $\varphi_C(\bar{u}, p)$, where $\bar{u} = (q_t, x, y, \hat{x}, \hat{y}, q_k)$ and $\bar{u}' = (q'_t, x', y', \hat{x}', \hat{y}', q'_k)$, such that $\varphi_E[T, \alpha; \bar{u}, \bar{u}'] = E$, and $\{\langle n\rangle \mid n \in \varphi_C[T, \alpha[\bar{a}/\bar{u}]; p]\} = C(\bar{a})$ for each $\bar{a} \in V$.

Let $\varphi_{\cong}(x, y) := \exists \bar{r}\, [\mathsf{lrec}_{\bar{u},\bar{u}',p}\ \varphi_E,\ \varphi_C]((0, x, y, x, y, 0), \bar{r})$, where $\bar{r}$ is a 5-tuple of number variables.[4] Let X be the relation defined by $\varphi_{\cong}$ in (T, α). By induction on $\mathrm{size}(v)$ it is easy to see that $(\bar{a}_{v,w}, \ell) \in X$ implies $T_v \cong T_w$. It remains to prove completeness:

▶ **Lemma 4.1.** *If $T_v \cong T_w$, then for all $\ell \geq \mathrm{size}(v)^5$ we have $(\bar{a}_{v,w}, \ell) \in X$.*

Proof. The proof is by induction on $\mathrm{size}(v)$. Suppose that $\mathrm{size}(v) = 1$ and $T_v \cong T_w$. Then $\mathrm{size}(w) = 1$ which implies that (v, w) is not easy. Furthermore, as v has no children in T, we know that $\bar{a}_{v,w}$ has no children in G and $C(\bar{a}_{v,w}) = \{0\}$. Hence, $(\bar{a}_{v,w}, \ell) \in X$ for all $\ell \geq 1 = \mathrm{size}(v)^5$.

Now suppose that $\mathrm{size}(v) = s + 1$ for some $s \geq 1$, and $T_v \cong T_w$. First note that (v, w) is not easy. Let $\ell \geq (s + 1)^5$. We show that $(\bar{a}_{v,w,\hat{v}}, \ell - 1) \in X$ for all children $\hat{v}$ of v, which implies $(\bar{a}_{v,w}, \ell) \in X$. Let $\hat{v}$ be a child of v in T. Since $T_v \cong T_w$, there is a child $\hat{w}$ of w of size $s' := \mathrm{size}(\hat{v})$ and a number $k \in [\#_{s'}(v)]$ such that

- $T_{\hat{v}} \cong T_{\hat{w}}$,
- there are exactly k children $\mathring{w}$ of w of size s' such that $T_{\hat{v}} \cong T_{\hat{w}}$, and
- there are exactly k children $\mathring{v}$ of v of size s' such that $T_{\mathring{v}} \cong T_{\hat{w}}$.

Pick such $\hat{w}$ and k.

Let us deal with the case $\#_{s'}(v) = 1$ first. In this case, $\bar{a}_{\hat{v},\hat{w}}$ is the only child of $\bar{a}_{v,w,\hat{v},\hat{w},k}$; moreover, $\bar{a}_{v,w,\hat{v},\hat{w},k}$ and $\bar{a}_{\hat{v},\hat{w}}$ have exactly one incoming edge each. Since $T_{\hat{v}} \cong T_{\hat{w}}$ and $\ell - 3 \geq (s')^5$, the induction hypothesis implies $(\bar{a}_{\hat{v},\hat{w}}, \ell - 3) \in X$. Consequently $(\bar{a}_{v,w,\hat{v}}, \ell - 1) \in X$.

In the following we assume $\#_{s'}(v) \geq 2$. Let $d := 3 \cdot \#_{s'}(v)^2$. Note that all vertices in Figure 1 except the type 0-vertices have exactly one incoming edge, and that the in-degree d' of a type 0-vertex $\bar{a}_{v',w'}$, where v', w' are children of v and w, respectively, of size s' is at most d, because it has incoming edges from

- vertices $\bar{a}_{v,w,v',w',k}$, where v and w are the (unique) parents of v' and w', respectively, and $k \in [\#_{s'}(v)]$;

[4] We use 0 as a constant, but clearly we can modify $\varphi_{\cong}$ to a formula that does not use the constant 0.

- vertices $\bar{a}^0_{v,w,v',\hat{w},k}$, where v, w, k are as above and $\hat{w}$ is a child of w of size s'; and
- vertices $\bar{a}^1_{v,w,\hat{v},w',k}$, where v, w, k are as above and $\hat{v}$ is a child of v of size s'.

Let $\ell' := \lfloor (\ell - 4)/d \rfloor$. Then

$$\ell' \geq \frac{\ell - d - 3}{d} \geq \frac{s^5}{d} + \frac{s^4}{d} - 2 \geq \frac{\#_{s'}(v)^5 \cdot (s')^5}{3 \cdot \#_{s'}(v)^2} + \frac{\#_{s'}(v)^4}{3 \cdot \#_{s'}(v)^2} - 2 \geq 2(s')^5 - 1 \geq (s')^5,$$

where for the second inequality we use $(s+1)^5 \geq s^5 + s^4$, for the third one we use $\#_{s'}(v) \cdot s' \leq s$, and for the fourth one we use $\#_{s'}(v) \geq 2$. Hence, by the induction hypothesis we have:

- $(\bar{a}_{\hat{v},\hat{w}}, \lfloor (\ell - 3)/d' \rfloor) \in X$ (note that $\lfloor (\ell - 3)/d' \rfloor \geq \ell'$).
- There are exactly k children $\mathring{w}$ of w of size s' with $(\bar{a}_{\hat{v},\mathring{w}}, \lfloor (\ell - 4)/d' \rfloor) \in X$ (note that $\lfloor (\ell - 4)/d' \rfloor \geq \ell'$), which implies $(\bar{a}^0_{v,w,\hat{v},\hat{w},k}, \ell - 3) \in X$.
- There are exactly k children $\mathring{v}$ of v of size s' with $(\bar{a}_{\mathring{v},\hat{w}}, \lfloor (\ell - 4)/d' \rfloor) \in X$, which implies that $(\bar{a}^1_{v,w,\hat{v},\hat{w},k}, \ell - 3) \in X$.

It follows immediately that $(\bar{a}_{v,w,\hat{v},\hat{w},k}, \ell - 2) \in X$, and therefore $(\bar{a}_{v,w,\hat{v}}, \ell - 1) \in X$. ◀

Finally, let $(v, w) \in V(T)^2$. Then we have $\text{size}(v)^5 \leq |V(T)|^5 \leq |N(T)|^{|\bar{r}|} - 1$, and therefore $T \models \varphi_{\cong}[v, w]$ iff $(\bar{a}_{v,w}, |N(T)|^{|\bar{r}|} - 1) \in X$ iff $T_v \cong T_w$.

4.2 Defining an Order on Directed Trees

Lindell's tree canonisation algorithm is based on a logspace-computable linear order on isomorphism classes of directed trees. We show that a slightly refined version of this order is LREC-definable.

Let T be a directed tree. For each $v \in V(T)$ let $\pi(v) := \big(\text{size}(v), \#_1(v), \ldots, \#_{\text{size}(v)-1}(v)\big)$ be the *profile* of v.[5] Let $\preceq$ be the total preorder on $V(T)$,[6] where $v \prec w$ whenever

1. $\pi(v) < \pi(w)$ lexicographically, or
2. $\pi(v) = \pi(w)$ and the following is true: Let $v_1, \ldots, v_k$ and $w_1, \ldots, w_k$ be the children of v and w, respectively, ordered such that $v_1 \preceq \cdots \preceq v_k$ and $w_1 \preceq \cdots \preceq w_k$. Then there is an $i \in [k]$ with $v_i \prec w_i$, and for all $j < i$ we have $v_j \preceq w_j$ and $w_j \preceq v_j$.

Note that $v \preceq w$ and $w \preceq v$ imply $T_v \cong T_w$. We show that $\preceq$ is LREC-definable.

To simplify the presentation, we again fix a directed tree T and an assignment α, and we assume that $|V(T)| \geq 4$.

We apply the lrec-operator to the following graph $G = (V, E)$ with labels $C(v) \subseteq \mathbb{N}$ for each $v \in V$. Let $V := N(T) \times V(T)^4 \times N(T)$. For each $(v, w) \in V(T)^2$, the vertex $\bar{a}_{v,w} = (0, v, w, v, w, 0)$ represents "$v \prec w$". If $\pi(v) < \pi(w)$, then $\bar{a}_{v,w}$ has no outgoing edges and $C(\bar{a}_{v,w}) = \{0\}$. If $\pi(v) > \pi(w)$, then $\bar{a}_{v,w}$ has no outgoing edges and $C(\bar{a}_{v,w}) = \emptyset$. Note that the relation "$\pi(v) \leq \pi(w)$" is LREC-definable.

Suppose that $\pi(v) = \pi(w)$. For all $t, u \in V(T)$ let $\theta_u(t)$ be the number of children u' of u with $T_{u'} \cong T_t$. Call a child $\hat{v}$ of v *good* if $\theta_v(\hat{v}) > \theta_w(\hat{v})$ and for all children v' of v with $\text{size}(v') < \text{size}(\hat{v})$ we have $\theta_v(v') = \theta_w(v')$. Then it is not hard to see that $v \prec w$ precisely if there is a good child $\hat{v}$ of v, a child $\hat{w}$ of w of size $s := \text{size}(\hat{v})$ and a $k \in [\#_s(v)]$ such that $\hat{v} \prec \hat{w}$, there are exactly k children $\mathring{w}$ of w of size s with $\mathring{w} \prec \hat{v}$, there are exactly k children $\mathring{v}$ of v of size s with $\mathring{v} \prec \hat{w}$ and $T_{\mathring{v}} \ncong T_{\hat{v}}$, and for all k children w' of w of size s with $w' \prec \hat{v}$ we have $\theta_v(w') = \theta_w(w')$. The "decision tree" in Figure 2 checks precisely these conditions.

Using the formula $\varphi_{\cong}$ from the previous section it is now straightforward to construct LREC[$\{E\}$]-formulae $\varphi_E(\bar{u}, \bar{u}')$ and $\varphi_C(\bar{u}, p)$ that define the edge relation E of G and the

[5] Lindell's order can be obtained by replacing $\pi(v)$ with $\pi'(v) := \big(\text{size}(v), \#\text{children of } v\big)$.

[6] That is, $\preceq$ is a preorder on $V(T)$ such that for all $v, w \in V(T)$ we have $v \preceq w$ or $w \preceq v$.

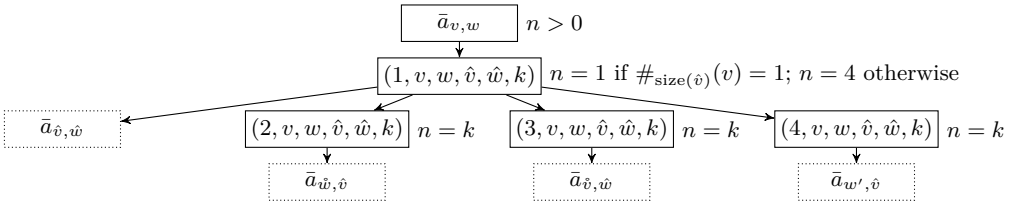


Figure 2 Gadget for deciding $v \prec w$ when $\pi(v) = \pi(w)$. Here, $\hat{v}$ ranges over good children of v; $\mathring{v}$ ranges over children of v of size $s := \mathrm{size}(v)$ and $T_{\mathring{v}} \ncong T_{\hat{v}}$; $\hat{w}, \mathring{w}$ range over children of w of size s; w' ranges over children of w of size s with $\theta_v(w') = \theta_w(w')$; and $k \in [\#_s(v)]$. The edges from $(2, v, w, \hat{v}, \hat{w}, k)$ to $(t, \dots)$ for $t \in \{2, 3, 4\}$ exist only if $\#_s(v) > 1$. Labels indicate which integers n belong to the set $C(\bar{a})$ labelling each vertex $\bar{a}$.

sets $C(\bar{a})$ for each $\bar{a} \in V$, where $\bar{u}$ and $\bar{u}'$ are as in the definition of $\varphi_{\cong}$. Let $\varphi_{\prec}(x, y) := \exists \bar{r}\, [\mathsf{lrec}_{\bar{u},\bar{u}',p}\ \varphi_E,\ \varphi_C]((0, x, y, x, y, 0), \bar{r})$, where $\bar{r}$ is a 5-tuple of number variables. Let X be the relation defined by $\varphi_{\prec}$ in (T, α). It is then possible to show by induction on $\mathrm{size}(v)$ that $(\bar{a}_{v,w}, \ell) \in X$ implies $v \prec w$ and that $v \prec w$ implies $(\bar{a}_{v,w}, \ell) \in X$ for all $\ell \geq \mathrm{size}(v)^5$. Hence, $T \models \varphi_{\prec}[v, w]$ iff $(\bar{a}_{v,w}, |N(T)|^{|\bar{r}|} - 1) \in X$ iff $v \prec w$.

4.3 Canonising Directed Trees

We now construct an LREC-formula $\gamma(p, q)$ such that for every directed tree T we have $T \cong ([|V(T)|], \gamma[T; p, q])$. Since DTC captures LOGSPACE on ordered structures [13] and a linear order is available on the number sort, we immediately obtain:

▶ **Theorem 4.2.** LREC *captures* LOGSPACE *on the class of directed trees.*

Since directed tree isomorphism is in LOGSPACE by Lindell's tree canonisation algorithm, but not TC+C-definable [5], we obtain:

▶ **Corollary 4.3.** LREC $\not\leq$ TC+C *on the class of all directed trees.*

We use l-recursion to define a set $X \subseteq V(T) \times N(T)^2$ (for simplicity, we omit the "resources" in the description) such that for every $v \in V(T)$ the set $X_v := \{(m, n) \in N(T)^2 \mid (v, m, n) \in X\}$ is the edge relation of an isomorphic copy $([|V(T_v)|], X_v)$ of T_v. Each vertex of T is numbered by its position in the preorder traversal sequence, e.g., the root is numbered 1, its first child v_1 is numbered 2, its second child v_2 is numbered $2 + \mathrm{size}(v_1)$, and so on.

To apply the lrec-operator, we define a graph $G = (V, E)$ with labels $C(v) \subseteq \mathbb{N}$ for each $v \in V$ as follows. Let $V := V(T) \times N(T)^2$, where $(v, m, n) \in V$ stands for "$(m, n) \in X_v$?". If v is a leaf, then X_v should be empty, so for all $m, n \in N(T)$ we let (v, m, n) have no outgoing edges and define $C((v, m, n)) := \emptyset$. Suppose that v is not a leaf and w is a child of v. Let S_w be the set of all children w' of v with $w' \prec w$, and let e_w be the number of children w' of v with $T_w \cong T_{w'}$. For each $i \in [0, e_w - 1]$, the set X_v will contain an edge from 1 to $p_{w,i} := 2 + \sum_{w' \in S_w} \mathrm{size}(w') + i \cdot \mathrm{size}(w)$, and the edges in $\{(p_{w,i} - 1 + m, p_{w,i} - 1 + n) \mid (m, n) \in X_w\}$. Hence we let $(v, 1, p_{w,i})$ have no outgoing edges and define $C((v, 1, p_{w,i})) := \{0\}$. Furthermore, for all $m, n \in N(T)$ and all $i < e_w$, we let $\bar{a} := (v, p_{w,i} - 1 + m, p_{w,i} - 1 + n)$ have an edge to (w, m, n) and define $C(\bar{a}) := \{1\}$.

It is now easy to construct LREC-formulae $\varphi_E(x_1, p_1, p_1', x_2, p_2, p_2')$ and $\varphi_C(x_1, p_1, p_1', q)$ that define the graph G and the labels $C(\cdot)$. Let

$$\gamma(p_1, p_2) := \exists x \exists r \big(\text{“}x \text{ is the root”} \wedge [\mathsf{lrec}_{(x_1,p_1,p_1'),(x_2,p_2,p_2'),q}\ \varphi_E,\ \varphi_C]((x, p_1, p_2), r)\big).$$

Noting that the in-degree of each vertex (v, m, n) is at most e_v, it is straightforward to show that γ defines an isomorphic copy of a directed tree.

5 Inexpressibility of Reachability in Undirected Graphs

In LREC it is not possible to define reachability in undirected graphs:

▶ **Theorem 5.1.** *There is no* LREC[$\{E\}$]*-formula* $\varphi(x, y)$ *such that for all undirected graphs* G *and all* $v, w \in V(G)$, $G \models \varphi[v, w]$ *iff there is a path from* v *to* w *in* G.

As an immediate corollary we obtain:

▶ **Corollary 5.2.** STC $\not\leq$ LREC

For the proof of Theorem 5.1, we consider the following undirected graphs G_n, for $n \geq 1$. Each graph G_n consists of $2 \cdot n^2$ vertices partitioned into *layers* $V_1^1, \ldots, V_n^1, V_1^2, \ldots, V_n^2$ with $|V_i^j| = n$, where every two vertices in consecutive layers V_i^j and V_{i+1}^j are connected by an edge, i.e., $E(G_n) = \{(v, w) \in V_i^j \times V_{i+1}^j \mid i \in [n-1],\, j \in [2]\}$. Vertices of G_n that belong to the same layer are called *siblings*. We show that reachability is not LREC-definable on the class $\mathcal{C}$ of all graphs that are isomorphic to G_n for some $n \geq 1$.

More precisely, we show that on $\mathcal{C}$ every LREC[$\{E\}$]-formula φ is equivalent to a formula in the *infinitary counting logic* $\mathcal{L}^*_{\infty\omega}(\mathbf{C})$ (see [19] or [20, Section 8.2]). Theorem 5.1 then immediately follows from the fact that every $\mathcal{L}^*_{\infty\omega}(\mathbf{C})$-formula without free number variables is Gaifman-local [19].

To construct an equivalent $\mathcal{L}^*_{\infty\omega}(\mathbf{C})$-formula we proceed by induction on the structure of the formula φ. The only interesting case is that of an LREC[$\{E\}$]-formula of the form $\varphi = [\mathsf{lrec}_{\bar{u}_1, \bar{u}_2, \bar{p}}\, \varphi_E,\ \varphi_C](\bar{w}, \bar{r})$. Let $\bar{v}_E$ be an enumeration of all variables in free(φ_E) that are not listed in $\bar{u}_1\bar{u}_2$ and let $\bar{v}_C$ be an enumeration of all variables in free(φ_C) that are not listed in $\bar{u}_1\bar{p}$. Let $n > |\bar{u}_1| + |\bar{v}_E| + 2$, and consider an assignment α in G_n. Further, let $V := G_n^{\bar{u}_1}$ and $E := \varphi_E[G_n, \alpha; \bar{u}_1, \bar{u}_2]$. For every $\bar{a} \in V$ and $\ell \in \mathbb{N}$, let $\mathcal{P}_{n,\ell}(\bar{a})$ be the set of all sequences $((\bar{a}_0, \ell_0), \ldots, (\bar{a}_m, \ell_m)) \in (V \times [0, \ell])^{m+1}$, where $m \in \mathbb{N}$, $(\bar{a}_0, \ell_0) = (\bar{a}, \ell)$, $(\bar{a}_0, \ldots, \bar{a}_m)$ is a path in (V, E), and $\ell_i = \lfloor (\ell_{i-1} - 1)/|E\bar{a}_i| \rfloor$ for each $i \in [m]$. The key property which enables us to construct a $\mathcal{L}^*_{\infty\omega}(\mathbf{C})$-formula equivalent to φ on $\mathcal{C}$ is:

▶ **Lemma 5.3.** *Let* $\bar{a} \in V$, $\ell \in \mathbb{N}$, *and* $((\bar{a}_0, \ell_0), \ldots, (\bar{a}_m, \ell_m)) \in \mathcal{P}_{n,\ell}(\bar{a})$. *Let* $\mathcal{I}$ *be the set of all* $i \in [m]$ *such that* $(\tilde{a}_{i-1} \cup \alpha(\tilde{v}_E)) \cap V(G_n) \neq (\tilde{a}_i \cup \alpha(\tilde{v}_E)) \cap V(G_n)$. *Then* $|\mathcal{I}|$ *is bounded by a constant that depends only on* φ.

The main insight for proving the lemma is that for every two siblings $b, b' \in V(G_n)$ there is an automorphism of G_n swapping b and b' and fixing all other vertices pointwise. Therefore, if $((\bar{a}_0, \ell_0), \ldots, (\bar{a}_m, \ell_m)) \in \mathcal{P}_{n,\ell}(\bar{a})$ and $i \in [m]$ is such that $\tilde{a}_{i-1} \cap V(G_n) \not\subseteq (\tilde{a}_i \cup \alpha(\tilde{v}_E)) \cap V(G_n)$, then for any $b \in \tilde{a}_{i-1} \cap V(G_n)$ with $b \notin \tilde{a}_i \cup \alpha(\tilde{v}_E)$, there is a linear number of siblings of b that do not occur in $\tilde{a}_i \cup \alpha(\tilde{v}_E) \cup \{b\}$, each leading to an incoming edge at $\bar{a}_i$. It is not hard to bound the number of all other $i \in \mathcal{I}$ by a constant.

6 An Extension of LREC

The previous section's Theorem 5.1 reveals that LREC is not closed under (first-order) logical reductions.[7] To remedy this weakness, we introduce the following extension LREC$_=$ of LREC.

[7] There is an FO-reduction that takes the graphs G_n, $n \geq 3$, considered in Section 5 to disjoint unions $\hat{G}_n$ of two undirected paths on n vertices each by identifying siblings. It is not hard to see that reachability

The idea is to admit a third formula $\varphi_=$ in the lrec-operator that generates an equivalence relation on the vertices of the graph defined by φ_E.

Let τ be a vocabulary. The set of all $\mathsf{LREC}_=[\tau]$-formulae is obtained from $\mathsf{LREC}[\tau]$ by replacing the rule for the lrec-operator from Section 3 as follows: If $\bar{u}, \bar{v}, \bar{w}$ are compatible tuples of variables, $\bar{p}, \bar{r}$ are non-empty tuples of number variables, and $\varphi_=$, φ_E and φ_C are $\mathsf{LREC}_=$-formulae, then the following is an $\mathsf{LREC}_=[\tau]$-formula:

$$\varphi := [\mathsf{lrec}_{\bar{u},\bar{v},\bar{p}}\ \varphi_=,\ \varphi_E,\ \varphi_C](\bar{w}, \bar{r}). \tag{3}$$

We let $\mathrm{free}(\varphi) := \big(\mathrm{free}(\varphi_=) \setminus (\tilde{u} \cup \tilde{v})\big) \cup \big(\mathrm{free}(\varphi_E) \setminus (\tilde{u} \cup \tilde{v})\big) \cup \big(\mathrm{free}(\varphi_C) \setminus (\tilde{u} \cup \tilde{p})\big) \cup \tilde{w} \cup \tilde{r}$.

To define the semantics of $\mathsf{LREC}_=[\tau]$-formulae φ of the form (3) let A be a τ-structure and α an assignment in A. Let $V_0 := A^{\bar{u}}$ and $E_0 := \varphi_E[A, \alpha; \bar{u}, \bar{v}]$. We define $\sim$ to be the reflexive, symmetric, transitive closure of the binary relation $\varphi_=[A, \alpha; \bar{u}, \bar{v}]$ over V_0. For every $\bar{a} \in V_0$ let $[\bar{a}]$ be the equivalence class of $\bar{a}$ with respect to $\sim$. Now consider the graph with vertex set $V := \{[\bar{a}] \mid \bar{a} \in V_0\}$ and edge set $E := \{[\bar{a}][\bar{b}] \in V^2 \mid \text{there are } \bar{a}' \in [\bar{a}],\ \bar{b}' \in [\bar{b}] \text{ with } \bar{a}'\bar{b}' \in E_0\}$. To every $[\bar{a}] \in V$ we assign the set $C([\bar{a}]) := \{\langle \bar{n} \rangle \mid \bar{n} \in \varphi_C[A, \alpha[\bar{a}'/\bar{u}]; \bar{p}], \bar{a}' \in [\bar{a}]\}$ of labels. Then the definition of X can be taken verbatim from Section 3. We let $(A, \alpha) \models \varphi$ if and only if $\big([\alpha(\bar{w})], \langle \alpha(\bar{r}) \rangle\big) \in X$.

As for LREC, the data complexity of $\mathsf{LREC}_=$ is in LOGSPACE and $\mathsf{LREC}_= \leq \mathsf{FP{+}C}$. Furthermore, $\mathsf{LREC}_=$ is closed under logical reductions.

The following example shows that undirected graph reachability is definable in $\mathsf{LREC}_=$. This does not involve an implementation of Reingold's algorithm in our logic, but just uses the observation that the computation of the equivalence relation $\sim$ boils down to the computation of undirected reachability.

▶ **Example 6.1** (Undirected reachability)**.** The following LREC-formula defines undirected graph reachability:

$$\varphi(s,t) \ :=\ [\mathsf{lrec}_{x,y,p}\ \varphi_=(x,y),\ \varphi_E(x,y),\ \varphi_C(x,p)](s,1),$$

where $\varphi_=(x,y) := E(x,y)$, $\varphi_E(x,y) := x \neq x$ and $\varphi_C(x,p) := x = t$. Let G be an undirected graph and α an assignment in G. Define V, E, C and the set X as above. Clearly, the set V consists of the connected components of G. Furthermore, the set E is empty since φ_E is unsatisfiable. Therefore, for all $v \in V(G)$ we have $([v], 1) \in X$ iff $0 \in C([v])$. The latter is true precisely if $\alpha(t) \in [v]$, i.e., if v and $\alpha(t)$ are in the same connected component of G. It follows that for all $v, w \in V(G)$ we have $G \models \varphi[v, w]$ if and only if v and w are in the same connected component of G, that is, if there is a path from v to w in G. ◀

It follows immediately from the previous example that $\mathsf{STC{+}C} \leq \mathsf{LREC}_=$. Actually, the containment is strict, because $\mathsf{LREC} \not\leq \mathsf{STC{+}C}$. Since trees can be made directed in STC+C, the results from Section 4 imply that $\mathsf{LREC}_=$ captures LOGSPACE on the class of all trees.

7 Capturing Logspace on Interval Graphs

We now prove that $\mathsf{LREC}_=$ captures LOGSPACE on the class of all interval graphs. The result is shown by describing an $\mathsf{LREC}_=$-definable canonisation procedure for interval graphs, which relies on a specific decomposition of graphs known as modular decomposition (first introduced

on the class of all graphs $\hat{G}_n$ is LREC-definable. Hence, if LREC was closed under FO-reductions, then reachability on the class of all graphs G_n would be LREC-definable, contradicting Theorem 5.1.

by Gallai [7]). It combines algorithmic techniques from [16] with the logical definability framework in [17]. The results in [17] are stated for fixed-point logic with counting only, but many of the results that are of interest for our construction hold in fact for STC+C. Parts of Sections 7.1 and 7.2 can be found in more detail in [18].

7.1 Definition of Interval Graphs and Basic Properties

▶ **Definition 7.1** (Interval graph)**.** Let $\mathcal{I}$ be a finite collection of closed intervals $I_i = [a_i, b_i] \subset \mathbb{N}$. The graph $G_\mathcal{I} = (V, E)$ has vertex set $V = \mathcal{I}$ and edge relation $I_iI_j \in E :\Leftrightarrow I_i \cap I_j \neq \emptyset$. $\mathcal{I}$ is called an *interval representation* of a graph G if $G \cong G_\mathcal{I}$, and a graph G is an *interval graph* if there is a collection of closed intervals $\mathcal{I}$ which is an interval representation of G.

A *clique* of a graph $G = (V, E)$ is a subset $C \subseteq V$ of the vertex set, such that the subgraph induced by C is complete. A maximal clique, or *max clique*, is a clique that is not properly contained in another clique. It is known [8, 22] that a graph G is an interval graph if and only if its max cliques can be brought into a linear order, so that each vertex of G is contained in consecutive max cliques. Let us denote the set of a graph's max cliques by $\mathcal{M}$. For canonisation it is essential to linearly order the max cliques of G.

Let $\mathrm{N}^\mathrm{c}(v)$ denote the closed neighbourhood of a vertex v, i.e. the set containing v and all vertices adjacent to v. The first lemma shows that the maximal cliques are FO-definable, as is the equivalence relation on V^2 of vertex pairs defining the same max clique.

▶ **Lemma 7.2** ([17], Lemma 4.1)**.** *Let G be an interval graph and let M be a max clique of G. Then there are vertices $u, v \in M$, not necessarily distinct, such that $M = \mathrm{N}^\mathrm{c}(u) \cap \mathrm{N}^\mathrm{c}(v)$.* ◀

The *span* of a vertex $v \in V$, denoted $\mathrm{span}(v)$, is the number of max cliques of G that v is contained in. Since equivalence classes can be counted in STC+C (Lemma 2.7. in [17]), the span of a vertex is STC+C-definable on the class of interval graphs.

7.2 Modular Decomposition Tree

A set W of vertices in a graph $G = (V, E)$ is a *module* if for all vertices $v \in V \setminus W$ either $\{v\} \times W \subseteq E$ or $(\{v\} \times W) \cap E = \emptyset$. The vertex set V and all vertex sets of size 1 are *trivial modules* by this definition. A module W is *proper* if $W \subset V$.

Let us call a vertex of G which is adjacent to all other vertices an *apex* of G. If G is a connected interval graph without an apex, then the complement graph of G is connected as well, and by [7] the set of maximal proper modules of G is a partition of G's vertex set. Thus, the set of proper modules $\mathcal{W}_G = \{W_1, \ldots, W_k\}$ of G which replaces every maximal proper module that is a subset of just one maximal clique by modules of size 1 for all contained vertices is also partition of G's vertex set. Each pair of modules W_i, W_j, $i \neq j$, is either completely connected or completely disconnected. Let $\sim_G$ be the equivalence relation corresponding to the partition $\mathcal{W}_G$, and let $L_G = G \,/\, \sim_G := (V \,/\, \sim_G, E_L)$, where $[u][v] \in E_L \Leftrightarrow \exists x \in [u], y \in [v]$ such that $xy \in E$. If G contains an apex, we let $\sim_G$ be the equivalence relation for which $x \sim_G y$ if and only if $x = y$ or $x, y \in V \setminus A$, where $A \subseteq V$ is the set of apices, and define L_G equivalently.

It is easy to see that L_G is an interval graph, that if A is a max clique of G, then $A_{L_G} := \{[v] \mid v \in A\}$ is a max clique of L_G, and that all max cliques of L_G are of this form. The following lemma gives further information about L_G.

▶ **Lemma 7.3.** *Let m be the number of max cliques of L_G.*

1. *There are* STC+C*-formulae* $\varphi_{\sim_G}$, φ_{L_G} *such that for all connected interval graphs,* $\varphi_{\sim_G}$ *defines the equivalence relation* $\sim_G$*, and* φ_{L_G} *the graph* L_G*.*
2. *If* $m > 1$*, then there exist exactly two linear orderings of* L_G*'s max cliques, each the reverse of the other. There is an* STC+C*-formula that given a max clique* A *of* G *defines the one with* A_{L_G} *appearing within the first* $\lfloor \frac{m}{2} \rfloor$ *max cliques of* L_G*.*
3. *There is an* STC+C*-formula that for all connected interval graphs* G *canonises* L_G*.*

According to the preceding lemma we can define an isomorphic copy $\mathcal{K}(L_G)$ of L_G on the number sort. What is left is to deal with the contents of the non-singular modules $W_{i_1}, \ldots, W_{i_l}$ of $\mathcal{W}_G$. If we continue decomposing the graphs $G[W_{i_1}], \ldots, G[W_{i_l}]$ inductively until we arrive at singular sets everywhere, we obtain a *modular decomposition* of G.

Let $P' = \{(M, n) \mid M \in \mathcal{M}, n \in [|V|]\}$. For each $(M, n) \in P'$ define $V_{M,n}$ as the set of vertices of the connected component of $G[\{v \in V \mid \text{span}(v) \leq n\}]$ which intersects M (if non-empty).

Let W be a non-singular maximal proper module. We define $\mathcal{C}$ to be the set of max cliques C such that $C \cap W \neq \emptyset$. It is immediate from the definition of a module W that $W = \bigcup_{C \in \mathcal{C}} V_{C,|\mathcal{C}|}$. Thus, for any $C \in \mathcal{C}$, the set $V_{C,|\mathcal{C}|}$ defines a component of W.

Let P be the set of those $(M, n) \in P'$ for which n is maximal among all n' with the property that $V_{M,n'} = V_{M,n}$ and which satisfy that $V_{M,n}$ is the connected component of a module occurring in the modular decomposition of G. Then P contains exactly all the components of modules occurring in the modular decomposition. There exists an STC+C-formula deciding whether (M, n) is in P.

Now we want to construct a coloured modular decomposition tree $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$. An illustration of the tree can be found in the full version. We will later need STC+C-definability of this coloured tree. Thus, notice that the tree's vertices are equivalence classes, which are STC+C definable. Also the edge relation and the colours are STC+C-definable (Lemma 7.3).

Let $V_\mathcal{T}$ be the union of the following sets:

- the set $\mathcal{V}$ of *component vertices* $v_{V_{M,n}}$, one for each set $V_{M,n}$ with $(M, n) \in P$,
- the set $\mathcal{A}$ of *arrangement vertices* $a_{\prec_Q, V_{M,n}}$ where $\prec_Q$ is the distinguished order on $L_{V_{M,n}}$'s max cliques if $\mathcal{K}(L_{V_{M,n}})$ is not order isomorphic under its two linear orderings, and if $\mathcal{K}(L_{V_{M,n}})$ is order isomorphic under its two linear orderings, then max clique Q identifies an order $\prec_Q$, namely, the order where $Q_{L_{V_{M,n}}}$ occurs first. (Q defines both orders if $Q_{L_{V_{M,n}}}$ is in the middle.)
- the set $\mathcal{S}$ of *module vertices* $s_{W_A, V_{M,n}}$ for which W_A is the module of $V_{M,n}$ that contains vertices of max clique A, and
- $\{s_V\}$, where s_V is a special vertex acting as the root of $\mathcal{T}$.

We colour the vertices in $\mathcal{V}$ by assigning to each $v_{V_{M,n}} \in \mathcal{V}$ the ordered graph $\mathcal{K}(L_{V_{M,n}})$. The vertices in $\mathcal{A}$ remain uncoloured and may therefore be exchanged by an automorphism of $\mathcal{T}$ whenever their subtrees are isomorphic. Each $s_{W_A, V_{M,n}} \in \mathcal{S}$ is coloured with the multiset of integers corresponding to the positions that the max clique $A_{L_{V_{M,n}}}$ takes in the orders of $L_{V_{M,n}}$.

The edge relation $E_\mathcal{T}$ of $\mathcal{T}$ is now defined in a straight-forward manner, with all edges directed away from the root s_V.

- s_V is connected to all $v_{V_{M,n}} \in \mathcal{V}$ with $n = |V|$.
- Each $v_{V_{M,n}} \in \mathcal{V}$ is connected to all vertices in $\mathcal{A}$ of the form $a_{\prec_Q, V_{M,n}}$ with $Q \cap V_{M,n} \neq \emptyset$.
- Each $a_{\prec_Q, V_{M,n}} \in \mathcal{A}$ is connected to all those $s_{W_A, V_{M,n}} \in \mathcal{S}$ so that $\prec_Q$ belongs to the set of orders of $L_{V_{M,n}}$ under which module $W_A \in L_{V_{M,n}}$ attains its minimal position,

that is, for every max clique Q that intersects with a non-singular module of $V_{M,n}$ vertex $a_{\prec_Q, V_{M,n}} \in \mathcal{A}$ is connected to $s_{W_Q, V_{M,n}} \in \mathcal{S}$.
- Every $s_{W_A, V_{M,n}} \in \mathcal{S}$ is connected to those $v_{V_{M',n'}} \in \mathcal{V}$ for which $V_{M',n'}$ is a connected component of the module W_A, that is, $s_{W_A, V_{M,n}} \in \mathcal{S}$ is connected to $v_{V_{A,n'}} \in \mathcal{V}$ with $n' = \max\{m < n \mid (V_{A,m}) \in P\}$.

The point of the arrangement vertices $\mathcal{A}$ is to ensure that the order of submodules is properly accounted for. If our modular tree did not have such a safeguard, exchanging modules in symmetric positions might give rise to a non-isomorphic graph, but it would not change the tree, so $\mathcal{T}$ would be useless for the task of distinguishing between these two graphs.

Lemma 7.4 below shows that our modular trees are a complete invariant of interval graphs, so modular trees can be used to tell whether two interval graphs are isomorphic.

▶ **Lemma 7.4** ([16], see full version for further remarks)**.** *Let G and H be interval graphs. If their modular trees are isomorphic, then so are G and H.* ◀

7.3 Canonisation

We can now make use of the STC+C-definable modular decomposition tree:

▶ **Lemma 7.5.** *Let $\theta_V(\bar{u})$, $\theta_\approx(\bar{u}, \bar{v})$, $\theta_E(\bar{u}, \bar{v})$ and $\theta_L(\bar{u}, \bar{q})$ be* STC+C*-formulae with $\bar{u}, \bar{v}$ compatible tuples and $\bar{q}$ a tuple of number variables, such that for all interval graphs G and assignments α, $\theta_\approx[G, \alpha; \bar{u}, \bar{v}]$ generates an equivalence relation $\approx$, $\theta_V[A, \alpha; \bar{u}] \,/\!\approx$ the vertices, $\theta_E[A, \alpha; \bar{u}, \bar{v}] \,/\!\approx$ the edge relation and $\theta_L[A, \alpha; \bar{u}, \bar{q}] \,/\!\approx$ the colours of the modular decomposition tree $\mathcal{T}_G$. Then there is an* LREC$_=$*-formula $\psi_{\preceq'}(\bar{u}, \bar{v})$ such that $\psi_{\preceq'}[A, \alpha; \bar{u}, \bar{v}] \,/\!\approx$ defines for all G a total preorder on the vertices of $\mathcal{T}_G$, which is more precisely, a linear order on the isomorphism classes of the (coloured) subtrees of $\mathcal{T}_G$ identified by its root vertices.*

Finally, we can use the modular decomposition tree and the total preorder on its vertices for canonisation. We use l-recursion on the modular decomposition tree, and as we have done for canonising trees we build the canon from the leaves to the root of the tree. Recursively, we construct the canon by first building the disjoint union of the canons of the components of submodules, then use the arrangement vertices to insert all submodules at the correct side and build the canon of the correspondent component of a module.

▶ Remark. It is possible to show that there is no LREC+TC$[\{E\}]$-sentence φ such that for all connected interval graphs G_1, G_2 we have $G_1 \uplus G_2 \models \varphi$ if and only if $G_1 \cong G_2$. The proof is based on similar ideas as the proof of Theorem 5.1.

8 Conclusion

We introduce the new logics LREC and LREC$_=$, extending first-order logic with counting by a recursion operator that can be evaluated in logarithmic space. By capturing LOGSPACE on trees and interval graphs, we obtain the first nontrivial descriptive characterisations of LOGSPACE on natural classes of unordered structures. It would be interesting to extend our results to further classes of structures such as the class of planar graphs or classes of graphs of bounded tree width.

The expressive power of LREC$_=$ is not yet well-understood. For example, it is an open question whether directed graph reachability is expressible in LREC$_=$, and even whether LREC$_=$ has the same expressive power as FP+C. (Of course assumptions from complexity theory indicate that the answer to both questions is negative.) It is also an open question whether reachability on undirected trees is expressible in plain LREC.

It is obvious that our capturing results can be transferred to nondeterministic logarithmic space NL by adding a transitive closure operator to the logic. However, it would be much nicer to have a natural "nondeterministic" variant of our limited recursion operator that allows it to express directed graph reachability and thus yields a logic that contains TC. We leave it as an open problem to find such an operator.

References

1 Martin Beaudry and Pierre McKenzie. Circuits, matrices, and nonassociative computation. *J. Comput. Syst. Sci.*, 50(3):441–455, 1995.

2 J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1992.

3 A. Chandra and D. Harel. Structure and complexity of relational queries. *J. Comput. Syst. Sci.*, 25:99–128, 1982.

4 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 1995.

5 K. Etessami and N. Immerman. Tree canonization and transitive closure. *Inf. Comput.*, 157:2–24, 2000.

6 R. Fagin. Generalized first–order spectra and polynomial–time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings 7*, pages 43–73, 1974.

7 T. Gallai. Transitiv orientierbare Graphen. *Acta Math. Hungar.*, 18(1-2):25–66, 1967.

8 P. C. Gilmore and A. J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canad. J. Math.*, 16:539–548, 1964.

9 E. Grädel, P.G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M.Y. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and Its Applications*. Springer-Verlag, 2007.

10 M. Grohe. Fixed-point definability and polynomial time on graphs with excluded minors. In *LICS*, 2010.

11 N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

12 N. Immerman. Expressibility as a complexity measure: results and directions. In *Structure in Complexity Theory Conference*, pages 194–202, 1987.

13 N. Immerman. Languages that capture complexity classes. *SIAM JoC*, 16:760–778, 1987.

14 N. Immerman. *Descriptive Complexity*. Springer-Verlag, 1999.

15 N. Immerman and E. Lander. Describing graphs: a first-order approach to graph canonization. In A. Selman, editor, *Complexity theory retrospective*. Springer-Verlag, 1990.

16 J. Köbler, S. Kuhnert, B. Laubner, and O. Verbitsky. Interval graphs: Canonical representation in logspace. In *ICALP (1)*, pages 384–395, 2010.

17 B. Laubner. Capturing polynomial time on interval graphs. In *LICS*, pages 199–208, 2010.

18 B. Laubner. *The Structure of Graphs and New Logics for the Characterization of Polynomial Time*. PhD thesis, Humboldt-Universität zu Berlin, 2011.

19 L. Libkin. Logics with counting and local properties. *ACM ToCL*, 1(1):33–59, 2000.

20 L. Libkin. *Elements of Finite Model Theory*. Springer-Verlag, 2004.

21 S. Lindell. A logspace algorithm for tree canonization. In *STOC*, pages 400–404, 1992.

22 R. H. Möhring. *Graphs and Order*, volume 147 of *NATO ASI Series C, Mathematical and Physical Sciences*, pages 41–102. D. Reidel, 1984.

23 M.Y. Vardi. The complexity of relational query languages. In *STOC*, pages 137–146, 1982.

The Lax Braided Structure of Streaming I/O*

Alan Jeffrey[1] and Julian Rathke[2]

1 **Alcatel–Lucent Bell Labs** ajeffrey@bell-labs.com
2 **University of Southampton** jr2@ecs.soton.ac.uk

Abstract

We investigate and implement a model of typed streaming I/O. Each type determines a language of traces analogous to regular expressions on strings, and programs are modelled by certain monotone functions on these traces. We show that sequential composition forms a lax braided monoid in the category of types and programs. This lax braided structure allows programs to be represented diagrammatically using Joyal and Street's string diagrams in 3D space. Monotone functions over traces cannot be executed efficiently, so we present an equivalent monoidal category of transducers. We demonstrate that transducers can be executed efficiently, theoretically by showing that programs with diagrams embedded in the plane can be executed in $O(1)$ space, and experimentally by an implementation in the Agda dependently typed functional language. Agda supports machine-assisted proof: we have mechanically verified that the transducer implementation and the I/O model form lax braided monoidal categories.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages; D.3.3 Language Constructs and Features

Keywords and phrases Semantics, categorical models, streaming I/O, Agda

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.292

1 Introduction

1.1 Semantics

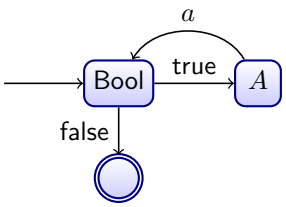
There are many models of streaming I/O, such as Kahn's dataflow networks [22] and Milner's [26] and Hoare's [16] process models. In these models, streams are *stateless*, for example a stream might be given the type Byte^ω, and a consumer is allowed to read a Byte from such a stream at any time.

To motivate the use of *stateful* streams, consider a typical Java program which consumes a steam of data:

```
Iterator<A> stream = ...;
while (stream.hasNext()) { A a = stream.next(); ... }
```

The contract for using an `Iterator<A>` stream is that `hasNext` is called, and depending on its value the stream is either terminated, or `next` can be called, and the stream's contract is back to its initial state:

* Support fo Julian Rathke provided by NSF Grant CCF-0916741



In Java, such contracts are enforced dynamically; they are enforced statically by systems of *typestates* [9]. Typestates are modelled as automata, and so come with the usual definition of *sequential composition*. We will write $T \,\&\, U$ for the sequential composition of T and U; a typical member is given by concatenating a member of T with a member of U. There is a matching notion of sequential composition of functions on traces, and so we investigate *monoidal categories*. There has been work on formal models of computation over stateful streams, notably *session* types [17] and *games* models [4, 18]. These models emphasise the *concurrent*, rather than sequential, composition of streams, for example in games models $T \otimes U$ is modelled by interleaving.

This paper provides the first categorical model for typed streaming I/O with a monoidal structure for sequential composition. We will show that this category has *lax braided* [8] structure (§2.5), and so has a dataflow presentation as *string diagrams* in three dimensions (§2.6). Braided monoidal categories are common in mathematical physics [5]; it is surprising that they also come up in the setting of streaming.

1.2 Pragmatics

This paper grew from an attempt to provide an I/O library for the Agda [1] dependently typed functional programming language. Since Agda compiles to Haskell [3], it is possible to link against Haskell's *lazy I/O* model. Unfortunately, lazy I/O does not respect Agda's semantics. Consider:

```
hello1 []       = putStr "Hello"          hello2 xs = putStr "Hello"
hello1 (x : xs) = putStr "Hello"
```

Agda includes a mechanized proof assistant, in which it is routine to prove that `hello1` and `hello2` are extensionally equivalent. Unfortunately, executing these programs using Haskell lazy I/O (`main = getContents >>= hello*`) results in `hello1` blocking waiting for input, and `hello2` immediately printing `"Hello"`. Kiselyov [23] has proposed an alternative to lazy I/O: the *iteratee* model; Millikin [25] has written a good introduction to the topic. Iteratees are similar to transducers [24] or resumptions [15]. Similar to this, we present a streamlined process model and show that processes are equivalent to functions over traces (§3.1).

We give a characterization of the *regular* functions on streams, which can be executed in $O(1)$ space. We show (§3.2) that regular programs can be presented as *planar* dataflow diagrams. We provide an implementation of processes as an Agda library (§3.3) that links processes against the Haskell I/O library, and verify experimentally that our implementation of a simple `wc` program runs in constant space. We have used Agda to mechanically validate many of the theorems in this paper. This verification is of the I/O library implementation, not just its model, and caught some corner case buffering bugs. This is the first mechanical verification of the categorical structure of an I/O library.

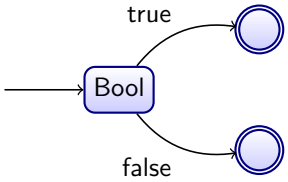
2 Semantics

2.1 Types

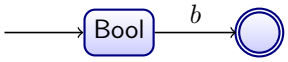
We model *stateful types* as the grammar:

$$T ::^{\nu}= I \mid \Sigma(a : A)\, T_a$$

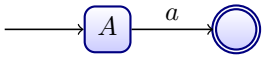
I is the *unit* type and $\Sigma(a : A)\, T_a$ is a *sum* type, where A is a set[1], and T is an A-indexed family of types. Types are defined *coinductively*, that is each type can be viewed as a (possibly infinitely deep and infinitely wide) tree, where each node is either: an I node, with no children, or a $\Sigma(A)$ node, with A-indexed children. We indicate that the grammar of types is to be interpreted coinductively by the annotation $::^{\nu}=$; we will annotate inductive grammars by $::^{\mu}=$. For example, the type: $\langle \mathsf{Bool} \rangle \stackrel{\mathsf{def}}{=} \Sigma(b : \mathsf{Bool})\, I$ has tree representation:



which can be viewed as the tree unfolding of the graph:

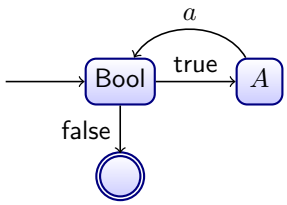


In general, the *character* type $\langle A \rangle$ is:



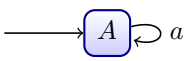
$$\langle A \rangle \stackrel{\mathsf{def}}{=} \Sigma(a : A)\, I$$

The *iterator* character type $\langle A \rangle^*$ is:



$$\langle A \rangle^* \stackrel{\mathsf{def}}{=} \Sigma(b : \mathsf{Bool})\, T_b$$
$$T_{\mathsf{true}} \stackrel{\mathsf{def}}{=} \Sigma(a : A)\, \langle A \rangle^*$$
$$T_{\mathsf{false}} \stackrel{\mathsf{def}}{=} I$$

The *stream* character type $\langle A \rangle^\omega$ is:



$$\langle A \rangle^\omega \stackrel{\mathsf{def}}{=} \Sigma(a : A)\, \langle A \rangle^\omega$$

As these examples show, types can be viewed as (potentially infinite state) automata, where the only acceptor is I.

▶ **Theorem 1.** *Types are in one-to-one correspondence with minimal deterministic automata where every acceptor is a sink state.*

[1] For readers who care about cardinality, let A range over a universe of small sets, and let the set of types be a large set.

2.2 Traces

Since types are automata, we can look at their languages. Define the transition relation on types and the language of a type:

$$\Sigma(a:A)\,T_a \xrightarrow{b} T_b \quad (b \in A) \qquad\qquad \mathcal{L}(T) = \{\, a_1 \cdots a_n \mid T \xrightarrow{a_1} \cdots \xrightarrow{a_n} I \,\}$$

Following [29], we will call the elements of this set *complete traces* of T. Equivalently, we can present the complete traces as a grammar:

$$t ::\overset{\mu}{=} \varepsilon \mid a \cdot t$$

together with a type judgement $t : \checkmark T$:

$$\frac{}{\varepsilon : \checkmark I} \qquad \frac{a \in A \quad t : \checkmark T_a}{a \cdot t : \checkmark \Sigma(a:A)\,T_a}$$

We can also define the language of (potentially *incomplete*) *traces* as:

$$\mathcal{T}(T) = \{\, a_1 \cdots a_n \mid \exists U \,.\, T \xrightarrow{a_1} \cdots \xrightarrow{a_n} U \,\}$$

or equivalently as a type judgement $t : T$:

$$\frac{}{\varepsilon : T} \qquad \frac{a \in A \quad t : T_a}{a \cdot t : \Sigma(a:A)\,T_a}$$

We are interested in incomplete traces, because we will view programs as functions from input traces to output traces. If we only recorded complete traces, then every program has an equivalent program which blocks waiting for its input to complete.

2.3 Categories

We have discussed our model of types as languages of traces, and now consider our model of programs as functions on traces. For any function $f : \mathcal{T}(T) \to \mathcal{T}(U)$, define:

- f is *monotone* whenever $t \leq u$ implies $f(t) \leq f(u)$, where $\leq$ is prefix order, and
- f *respects completion* whenever t is complete implies $f(t)$ is complete.

Monotonicity is a standard requirement for trace models, for example [22], as it expresses that a program must commit to its output. Respecting completion is a termination property. This leads us to our first category of functions over traces. **Tr** is the category with:

- Objects are types.
- Morphisms $f : T \to U$ are monotone functions $f : \mathcal{T}(T) \to \mathcal{T}(U)$ which respect completion.
- Identity and composition are as expected.

It turns out that we will use two other conditions on functions in Sections 2.4 and 2.5. Define:

- f *reflects completion* whenever $f(t)$ is complete implies t is complete, and
- f is *strict* whenever $f(\varepsilon) = \varepsilon$.

We can then define three subcategories of **Tr**, all with the same objects:

- in **RTr**, morphisms reflect completion,
- in **STr**, morphisms are strict, and
- in **RSTr**, morphisms reflect completion and are strict.

It is routine to verify that identity and composition preserve monotonicity, respecting completion, reflecting completion, and strictness, and so form categories.

▶ **Theorem 2.** **Tr**, **RTr**, **STr** *and* **RSTr** *are categories.*

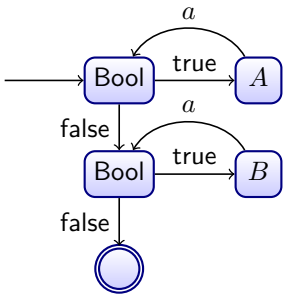
Proof. Mechanically verified [19]. ◀

2.4 Monoidal structure

Since types are automata, they come equipped with a monoidal action: *sequential composition*. We define the type $T \,\&\, U$ by tree substitution :

$$I \,\&\, U \stackrel{\mathsf{def}}{=} U \qquad (\Sigma(a:A)\, T_a) \,\&\, U \stackrel{\mathsf{def}}{=} \Sigma(a:A)\, (T_a \,\&\, U)$$

This is the usual definition of composition of automata, replacing any moves to the acceptor in T by a move to the initial state of U, for example:



$\langle A \rangle^* \,\&\, \langle B \rangle^*$

It is easy to check that on types, & forms a monoid with unit I. To define the action of & on morphisms, we first define concatenation of traces: given $t : T$ and $u : U$, we define $t \frown u : T \,\&\, U$. If t is complete, the definition is as expected:

$$\varepsilon \frown u \stackrel{\mathsf{def}}{=} u \qquad\qquad (a \cdot t) \frown u \stackrel{\mathsf{def}}{=} a \cdot (t \frown u)$$

We cannot, however, use this definition when t is incomplete, as it does not typecheck; instead we define:

$$t \frown u \stackrel{\mathsf{def}}{=} t \text{ when } t \text{ is incomplete}$$

Given $t : T \,\&\, U$, we define $\mathsf{front}_T(t) : T$ and $\mathsf{back}_T(t) : U$ as:

$$\begin{aligned} \mathsf{front}_I(t) &\stackrel{\mathsf{def}}{=} \varepsilon & \mathsf{back}_I(t) &\stackrel{\mathsf{def}}{=} t \\ \mathsf{front}_{\Sigma(a:A)\, T_a}(\varepsilon) &\stackrel{\mathsf{def}}{=} \varepsilon & \mathsf{back}_{\Sigma(a:A)\, T_a}(\varepsilon) &\stackrel{\mathsf{def}}{=} \varepsilon \\ \mathsf{front}_{\Sigma(a:A)\, T_a}(a \cdot t) &\stackrel{\mathsf{def}}{=} a \cdot \mathsf{front}_{T_a}(t) & \mathsf{back}_{\Sigma(a:A)\, T_a}(a \cdot t) &\stackrel{\mathsf{def}}{=} \mathsf{back}_{T_a}(t) \end{aligned}$$

We will often elide the types from front and back. From concatenation and projection, we can define the action of & on morphisms. Given $f : T \to U$ and $g : T' \to U'$, define $f \,\&\, g : T \,\&\, T' \to U \,\&\, U'$ as:

$$(f \,\&\, g)(t) \stackrel{\mathsf{def}}{=} f(\mathsf{front}(t)) \frown g(\mathsf{back}(t))$$

Unfortunately, this is *not* a monoid on **Tr**, as it is not a functor, for which we would need:

$$(f_1; g_1) \,\&\, (f_2; g_2) = (f_1 \,\&\, f_2); (g_1 \,\&\, g_2)$$

This fails for morphisms (of type $\langle \mathsf{Bool} \rangle \to \langle \mathsf{Bool} \rangle$):

$$f_1(t) \stackrel{\mathsf{def}}{=} g_2(t) \stackrel{\mathsf{def}}{=} t \qquad f_2(t) \stackrel{\mathsf{def}}{=} g_1(t) \stackrel{\mathsf{def}}{=} \mathsf{true} \cdot \varepsilon \qquad \mathit{lhs}(\varepsilon) = \mathsf{true} \cdot \mathsf{true} \cdot \varepsilon \neq \mathsf{true} \cdot \varepsilon = \mathit{rhs}(\varepsilon)$$

To show functoriality, it is sufficient to show:

- g_1 reflects completion, or
- f_1 respects completion, and f_2 is strict.

▶ **Theorem 3.** **RTr**, **STr** *and* **RSTr** *are monoidal categories.*

Proof. Mechanically verified [19]. ◀

2.5 Lax braided structure

We can define a family of morphisms:

$$\mathsf{swap} : T \,\&\, U \to U \,\&\, T \qquad \mathsf{swap}(t) \stackrel{\mathsf{def}}{=} \mathsf{back}(t) \frown \mathsf{front}(t)$$

Unfortunately, swap does *not* form a symmetry, as this would require $\mathsf{swap}(\mathsf{swap}(t)) = t$, which is only true (for non-I types) when t is complete:

$$\mathsf{swap}(\mathsf{swap}(t)) = \varepsilon \text{ when } t \text{ is incomplete}$$

The problem is that swap has to *buffer* the front of the input until the the input is complete. We will discuss this form of space usage in Section 3.2.

We have shown that swap is not a symmetry, and in fact it is it not even an isomorphism. For example, for $\mathsf{swap} : \langle \mathsf{Bool} \rangle \,\&\, U \to U \,\&\, \langle \mathsf{Bool} \rangle$ we have, for any f:

$$f(\mathsf{swap}(\mathsf{true} \cdot \varepsilon)) = f(\varepsilon) = f(\mathsf{swap}(\mathsf{false} \cdot \varepsilon))$$

and so f cannot be the inverse of swap.

Categories in which swap is an isomorphism have been studied in depth: they are *braided monoidal categories*, and have applications in mathematical physics, as surveyed, for example, by Baez and Stay [5]. Braided monoidal categories can be regarded as the categorical version of *braid groups*, where every braiding has an inverse. *Positive braid monoids* [10] drop this requirement; their categorical equivalent, *lax braided monoidal categories* have only recently been investigated by Day *et al.* [8], see Figures 2–3.

We will see that **RSTr** is a lax braided monoidal category. Unfortunately, **RTr** and **STr** are *not* lax braided, as swap is not natural, for which we need:

$$(f \,\&\, g); \mathsf{swap} = \mathsf{swap}; (g \,\&\, f)$$

To see that reflecting completion without strictness is not enough to guarantee naturality, consider:

$$f(t) \stackrel{\mathsf{def}}{=} t \qquad g(t) \stackrel{\mathsf{def}}{=} a \cdot t \qquad \mathit{lhs}(\varepsilon) = \varepsilon \neq a \cdot \varepsilon = \mathit{rhs}(\varepsilon)$$

To see that strictness without reflecting completion is not enough, consider, for any complete $t \neq \varepsilon$, incomplete $u \neq \varepsilon$ and complete s:

$$f(t) \stackrel{\mathsf{def}}{=} t \qquad g(t) \stackrel{\mathsf{def}}{=} \begin{cases} \varepsilon & \text{if } t = \varepsilon \\ s & \text{otherwise} \end{cases} \qquad \mathit{lhs}(t \frown u) = s \frown t \neq s = \mathit{rhs}(t \frown u)$$

Naturality of swap can be shown when f respects completion, and g is strict and reflects completion.

▶ **Theorem 4.** **RSTr** *is a lax braided monoidal category.*

Proof. Mechanically verified [19]. ◀

2.6 String diagrams

Braided monoidal categories have an associated graphical language of *string diagrams*, as shown by Joyal and Street [20]. String diagrams have been used in various guises for quite some time now and we refer the reader to Baez and Stay [5] or Selinger [27] up to date surveys of graphical languages.

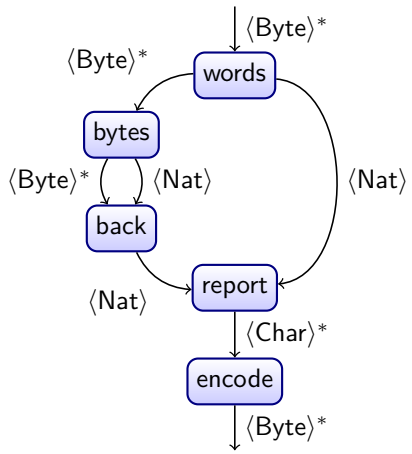


Figure 1 Example dataflow diagram.

Such diagrams are often used to represent the *dataflow* of programs, for example in Kahn networks [22]. An example dataflow program is shown in Figure 1: a simple `wc` program, which counts the number of bytes and the number of words in an input stream. (We draw diagrams flowing from top to bottom in line with Baez and Stay rather than Selinger).

The categorical structure of such dataflow diagrams is well-known: they form a symmetric monoidal category, with combinators shown in Figure 2. It is routine to verify that dataflow graphs up to isomorphism satisfy the properties given in Figures 3 and 4, which are the defining equations of a (strict) symmetric monoidal category. Joyal and Street have shown that not only are these equations sound, but they are also complete for graph isomorphism [20, Thm. 2.3], as discussed by Selinger [27, Thm. 3.12].

In a braided monoidal category, Figure 4 is replaced by Figure 5. This is accompanied by a matching change in the interpretation of diagrams; rather than graph isomorphism, we consider equivalence in three dimensional space. It is routine to verify that string diagrams up to isotopy[2] satisfy the properties given in Figures 3 and 5, which are the defining equations of a (strict) braided monoidal category. Again, Joyal and Street have shown that not only are these equations sound, but they are also complete for isotopy [20, Thm. 3.7], as discussed by Selinger [27, Thm. 3.7].

In string diagrams, swap is interpreted as a left-over-right crossing, and so has an inverse right-over-left crossing. In a lax braided monoidal category, the requirement that swap has an inverse is dropped, and we are left with the equations presented in Figure 3. This provides us with a graphical language of a lax braided monoidal category: we conjecture that this graphical language is sound and complete.

▶ Conjecture 2.1. A well-formed equation between morphisms in the language of lax braided monoidal categories follows from the axioms of lax braided monoidal categories if and only if it holds in the graphical language up to isotopy in 3 dimensions.

Proof sketch. Given Joyal and Street's results, it is enough to show that the equational theory of a braided monoidal category is a conservative extension of the equational theory

[2] More precisely, progressive isotopy of smooth string diagrams in three dimensions, see Joyal and Street [20].

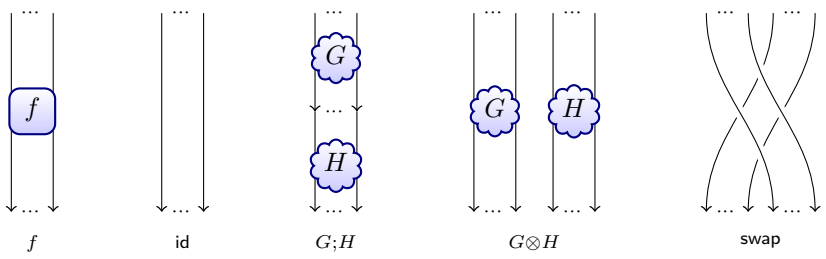


Figure 2 Combinators of a (strict) symmetric monoidal category.

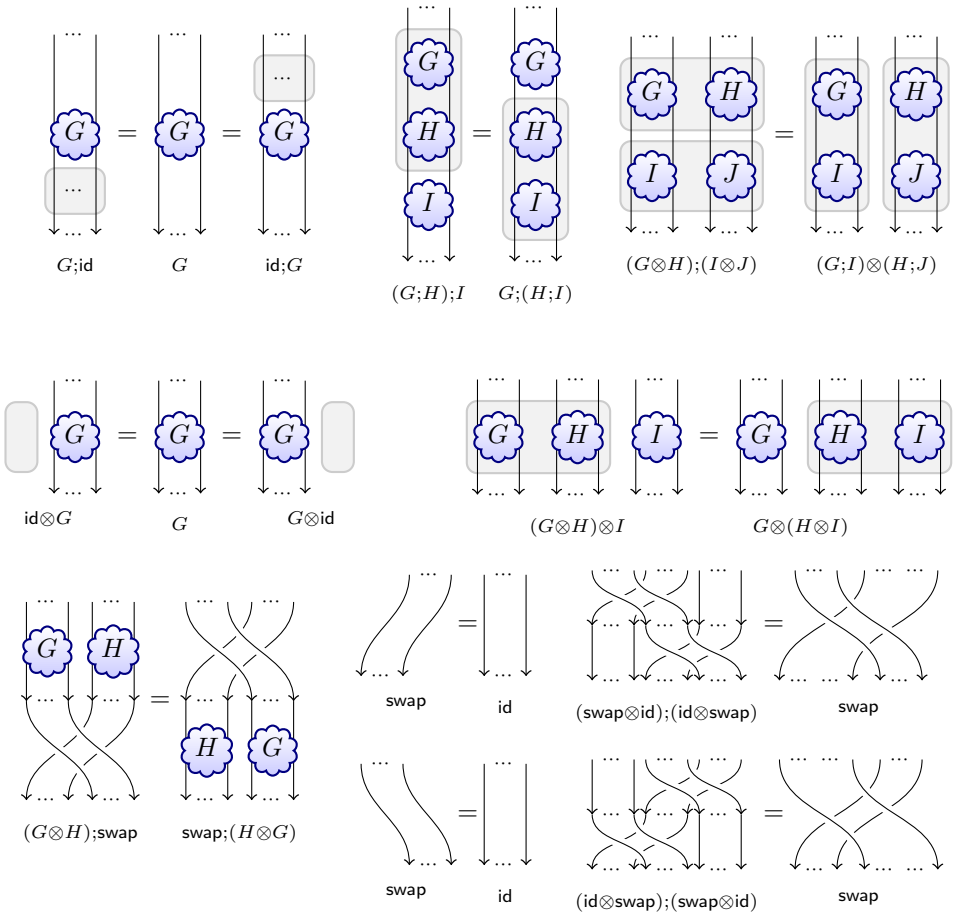


Figure 3 The equations of a (strict) lax braided monoidal category

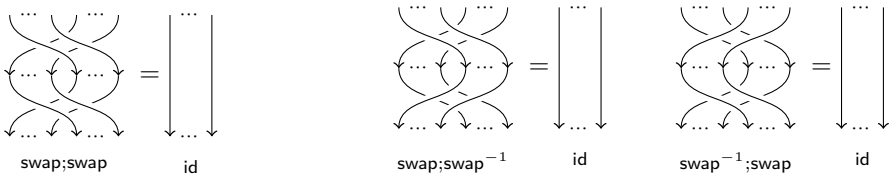


Figure 4 Symmetry

Figure 5 Braiding

of a lax braided monoidal category. That is, given any two morphisms f and g in the free lax braided monoidal category over a given signature, if $f =_B g$ then $f =_L g$ (where $=_B$ is the theory of a braid, and $=_L$ is the theory of a lax braid). In the case without generating morphisms, this problem collapses to the case of showing that the free positive braid monoid embeds into the free braid group, which was shown by Garside [10][3].

Let w range over *wiring morphisms* (that is, morphisms without generators) and p range over *planar morphisms* (that is, morphisms without swap). Define a *stratified* morphism to be one of the form: $w_0; p_1; w_1; \ldots; p_n; w_n$ such that if $p_i; w_i =_L w; f$ for some w, then $w =_L \mathsf{id}$, and if $w_i; p_{i+1} =_L p; f$ for some p, then $p =_L \mathsf{id}$. Now, if we can show that:

- every morphism can be stratified up to $=_L$,
- stratified terms are a normal form for $=_B$, and
- Conjecture 3.4 of Selinger [27],

then we can prove our conjecture. Assume morphisms f and g in the free lax braided monoidal category, such that $f =_B g$. Stratify each of them to get $f =_L w_0; f_1; w_1; \ldots; f_n; w_n$ and $g =_L v_0; g_1; v_1; \ldots; g_b; w_n$. Since these are normal forms for $=_B$, we have: $v_i =_B w_i$ and $f_i =_B g_i$. Each $v_i =_L w_i$ from Garside [10]. Since $f_i =_B g_i$, we have that f_i and g_i are isotopic as string diagrams in three dimensions, so Selinger's conjecture implies $f_i =_L g_i$. The result then follows. ◀

We leave the full proof of this conjecture as future work. If this conjecture is true, it provides a powerful, and unexpected, proof technique for equivalence of streaming programs: draw the dataflow diagrams for the programs as string diagrams in three dimensions, and check that an isotopy exists. Isotopies can often be checked "by eye", so this technique would allow many routine rewiring steps in a proof to be elided.

3 Pragmatics

3.1 Processes

One of the major drawbacks of functions on partial traces as a model of streaming I/O is that they cannot easily be executed directly. Imagine an execution of f after receiving input t: when a new input symbol a arrives, we need to know the matching output, so we have to apply f to the new history $t \cdot a$. This is inefficient in both time (potentially $O(n^2)$ rather than $O(n)$) and space (potentially $O(n)$ rather than $O(1)$). Delimited call/cc [28] would avoid this, at the cost of sophisticated language features.

Therefore, as a move towards an implementation we find it useful to introduce a syntactic representation of programs as a small language of *transducer processes*:

$$S ::\overset{\nu}{=} \mathsf{inp}(a : A)\, P_a \mid \mathsf{done} \qquad P ::\overset{\mu}{=} S \mid \mathsf{out}\, a\, P$$

In this definition:

- $\mathsf{inp}(a : A)\, P_a$ is an *input* process, A is a set, and P is an A-indexed family of processes,
- done is the *terminated* process, and
- $\mathsf{out}\, a\, P$ is an *output* process.

Notice that the two levels of grammar define *strict* processes S and (*lazy*) processes P. Laziness here refers to not being reliant on an input in order to generate output. Also note

[3] Thanks to Ross Street for discussions on this topic, and for pointing us to Garside's work.

the coinductive annotation on the grammar of strict processes, and the inductive annotation on P, which ensures that there are no infinite sequences of output actions.

The type rules for processes are given coinductively:

$$\frac{}{\mathsf{done} : I \to I} \qquad \frac{P_a : T_a \to U \text{ for all } a \in A}{\mathsf{inp}(a:A)\, P_a : \Sigma(a:A)\, T_a \to U} \qquad \frac{a : A \qquad P : T \to U_a}{\mathsf{out}\, a\, P : T \to \Sigma(a:A)\, U_a}$$

Note that since there are no infinite sequences of output actions, well=typed processes respect completion.

We can define[4] the operation of composition on processes $\gg$ as (in order):

$$\begin{aligned} P \gg \mathsf{out}\, a\, Q &\stackrel{\mathsf{def}}{=} \mathsf{out}\, a\, (P \gg Q) \\ \mathsf{inp}(a:A)\, P_a \gg Q &\stackrel{\mathsf{def}}{=} \mathsf{inp}(a:A)\, (P_a \gg Q) \\ \mathsf{out}\, a\, P \gg \mathsf{inp}(b:B)\, Q_b &\stackrel{\mathsf{def}}{=} P \gg Q_a \\ \mathsf{done} \gg Q &\stackrel{\mathsf{def}}{=} Q \\ P \gg \mathsf{done} &\stackrel{\mathsf{def}}{=} P \end{aligned}$$

The identity processes for this composition are:

$$\mathsf{id}_I = \mathsf{done} \qquad \mathsf{id}_{\Sigma(a:A)\, T_a} = \mathsf{inp}(a:A)\, \mathsf{out}\, a\, \mathsf{id}_{T_a}$$

This leads us to a category of processes. **Pr** has:

- Objects are types.
- Morphisms $P : T \to U$ are processes.
- Identity and composition are id and $\gg$.

We have already defined the strict processes. A process *reflects completion* when it can be typed with an additional side-condition on the type rule for input:

$$\frac{P_a : T_a \to U \text{ for all } a \in A}{\mathsf{inp}(a:A)\, P_a : \Sigma(a:A)\, T_a \to U} (U \neq I)$$

We can then define three subcategories of **Pr**, all with the same objects:

- in **RPr**, processes reflect completion,
- in **SPr**, processes are strict, and
- in **RSPr**, processes reflect completion and are strict.

We can define a sequential composition operator $P \,\&\, Q$ on typed processes as (in order):

$$\begin{aligned} P \,\&\, \mathsf{out}\, b\, Q &\stackrel{\mathsf{def}}{=} \mathsf{out}\, b\, (P \,\&\, Q) \quad \text{if } P : T \to I \\ \mathsf{done} \,\&\, Q &\stackrel{\mathsf{def}}{=} Q \\ \mathsf{inp}(a:A)\, P_a \,\&\, Q &\stackrel{\mathsf{def}}{=} \mathsf{inp}(a:A)\, (P_a \,\&\, Q) \\ \mathsf{out}\, a\, P \,\&\, Q &\stackrel{\mathsf{def}}{=} \mathsf{out}\, a\, (P \,\&\, Q) \end{aligned}$$

This definition is straightforward except for the first clause in which a process that has completed its output will not block immediate output from Q.

[4] Some care is required to ensure that this definition is well formed, since it uses a mix of induction and coinduction. This has been mechanically verified [19].

The lax braided structure on processes is given by the $\mathsf{swap}_{T,U}$ process defined (in order):

$$\begin{aligned}
\mathsf{swap}_{I,U} &\stackrel{\mathsf{def}}{=} \mathsf{id}_U \\
\mathsf{swap}_{T,I} &\stackrel{\mathsf{def}}{=} \mathsf{id}_T \\
\mathsf{swap}_{T,U} &\stackrel{\mathsf{def}}{=} \mathsf{swap}_{T,U}(\varepsilon) \qquad\qquad \mathsf{out}\,\varepsilon\,P \stackrel{\mathsf{def}}{=} P \\
\mathsf{swap}_{I,I}(u) &\stackrel{\mathsf{def}}{=} \mathsf{out}\,u\,\mathsf{done} \qquad\qquad \mathsf{out}\,(u \cdot a)\,P \stackrel{\mathsf{def}}{=} \mathsf{out}\,u\,\mathsf{out}\,a\,P \\
\mathsf{swap}_{I,\Sigma(a:A)\,U_a}(u) &\stackrel{\mathsf{def}}{=} \mathsf{inp}(a:A)\,\mathsf{out}\,a\,\mathsf{swap}_{I,U_a} \\
\mathsf{swap}_{\Sigma(a:A)\,T_a,U}(u) &\stackrel{\mathsf{def}}{=} \mathsf{inp}(a:A)\,\mathsf{swap}_{T_a,U}(u \cdot a)
\end{aligned}$$

This process explicitly maintains a buffer u of actions which are output after its input has completed.

▶ **Theorem 5.**

1. **Pr** *is a category.*
2. **RPr** *and* **SPr** *are monoidal categories.*
3. **RSPr** *is a lax braided monoidal category.*

Proof. Mechanically verified [19]. ◀

In order to show that our transducer processes accurately represent our model, we show equivalences of categories. Given a morphism $P : T \to U$ in **Pr** we give a morphism $[\![P]\!] : T \to U$ in **Tr** as follows:

$$\begin{aligned}
[\![\mathsf{done}]\!](\varepsilon) &\stackrel{\mathsf{def}}{=} \varepsilon \\
[\![\mathsf{out}\,a\,P]\!](t) &\stackrel{\mathsf{def}}{=} a \cdot [\![P]\!](t) \\
[\![\mathsf{inp}(a:A)\,P_a]\!](\varepsilon) &\stackrel{\mathsf{def}}{=} \varepsilon \\
[\![\mathsf{inp}(a:A)\,P_a]\!](b \cdot t) &\stackrel{\mathsf{def}}{=} [\![P_b]\!](t)
\end{aligned}$$

On traces, define $t \xrightarrow{u} t'$ as:

$$\frac{}{t \xrightarrow{\varepsilon} t} \qquad \frac{t \xrightarrow{u} t'}{a \cdot t \xrightarrow{a \cdot u} t'}$$

that is, whenever t can be partitioned into a prefix u and suffix t'. On morphisms, define:

$$f \xrightarrow{t/u} f' \text{ whenever } s \xrightarrow{t} s' \text{ implies } f(s) \xrightarrow{u} f'(s')$$

This allows us to view morphisms as (possibly infinite state) *transducers* [24]: f responds to input t by producing output u and changing state to f'. Given a strict morphism $f : T \to U$, define the strict process $(\!|f|\!)^s_T : T \to U$ as:

$$\begin{aligned}
(\!|f|\!)^s_I &\stackrel{\mathsf{def}}{=} \mathsf{done} \\
(\!|f|\!)^s_{\Sigma(a:A)\,T_a} &\stackrel{\mathsf{def}}{=} \mathsf{inp}(a:A)\,\mathsf{out}\,t\,(\!|f'|\!)^s_T \text{ where } f \xrightarrow{a/t} f'
\end{aligned}$$

Given a morphism $f : T \to U$, define the process $(\!|f|\!)_T : T \to U$ as:

$$(\!|f|\!)_T \stackrel{\mathsf{def}}{=} \mathsf{out}\,t\,(\!|f'|\!)^s_T \text{ where } f \xrightarrow{\varepsilon/t} f'$$

We can show that $(\!|\cdot|\!)$ and $[\![\cdot]\!]$ are inverses and respect categorical structure, and so form equivalences.

▶ **Theorem 6.** $(\!|\cdot|\!)$ *and* $[\![\cdot]\!]$ *form equivalences:*
1. **Tr** *and* **Pr** *as categories.*
2. **RTr** *and* **RPr** *as monoidal categories.*
3. **STr** *and* **SPr** *as monoidal categories.*
4. **RSTr** *and* **RSPr** *as lax braided monoidal categories.*

Proof. Mechanically verified [19]. ◀

3.2 Space usage

In Section 2.5, we discussed the fact that swap introduces buffering: we will now discuss space usage more formally.

An *online algorithm* is one which can be implemented by a multi-tape Turing Machine where only rightward moves are allowed on the input and output tapes. The space usage of such an implementation is the space usage of the scratch tapes: for example the identity function is considered to be in $O(1)$ space. It is routine to show that if f and g can be implemented in $O(1)$ space, then so can $f;g$ and $f \,\&\, g$. The only constructor of a lax braided monoidal category to introduce $O(n)$ space usage is swap.

Another way to characterize the space usage is by analogy with regular trees. In our setting, we are interested in the infinite trees generated by the coinductive definition of types. A type is *regular* whenever: $\{\, T' \mid T \xrightarrow{t} T' \,\}$ is finite and a morphism is *regular* whenever: $\{\, f' \mid f \xrightarrow{t/u} f' \,\}$ is finite It is routine to see that regular types (resp. morphisms) can be implemented as deterministic finite-state automata (resp. sequential finite-state transducers [24]), and so can be executed in $O(1)$ space (provided the alphabet can be represented in $O(1)$ space).

The identity function is regular, and regularity of morphisms is preserved by $f;g$, and $f \,\&\, g$, so we can form the subcategory (resp. strict monoidal subcategory) of **Tr** (resp. **RTr**, **STr** and **RSTr**) where morphisms are regular.

We can now show that swap is irregular. Consider the type $N \stackrel{\mathsf{def}}{=} 1.N + 0.I$ whose complete traces are of the form 1^n0, and whose incomplete traces are of the form 1^n. Now, for any m and n, if (at type $N \,\&\, N$):

$$\mathsf{swap} \xrightarrow{1^m/\varepsilon} f \qquad \mathsf{swap} \xrightarrow{1^n/\varepsilon} f$$

then we must have: $01^m0 = \mathsf{swap}(1^m00) = f(00) = \mathsf{swap}(1^n00) = 01^n0$ and hence $m = n$. Thus there must be infinitely many such f, and so swap is irregular. This argument is essentially a replay of the Pumping Lemma in the setting of transducers rather than automata.

Since any plane dataflow graph can be expressed without swap [27], this gives a surprising method for showing that a function can be implemented in $O(1)$ space: check that its dataflow diagram is embedded in a plane. Moreover, if Conjecture 2.1 is true, then this means that planarity only has to hold up to isotopy in three dimensions.

▶ **Theorem 7.** *Any plane dataflow diagram whose generating morphisms can be implemented in online* $O(1)$ *space determines a morphism which can be implemented in online* $O(1)$ *space.*

3.3 Implementation

Agda [1] is a dependently typed functional programming language, which supports mechanical theorem proving. Its core language is similar to that of Coq [2], although it does not

have a separate language of tactics. It has a compiler to Haskell [3], and a foreign function interface, which allows calls out to Haskell code.

The transducer process language and type system discussed in Section 3.1 is implemented as an Agda library. It is linked against the Haskell I/O monad, and compiles to simple pipes from standard input to standard output. The implementation allows us to verify experimentally that regular programs run in constant space, for example the `wc` program from Figure 1 can word count 29Mb of data (1M lines of XML) in 120k heap, with 22k live:

```
head -1000000 h_sapiens.xml | WC +RTS -A50k -M120k -s
29294872 1339150
  16,796,509,664 bytes allocated in the heap
     477,860,144 bytes copied during GC
          22,112 bytes maximum residency (1 sample(s))
          32,512 bytes maximum slop
```

Many of the theorems in this paper have been mechanically verified, which shows correctness not just of the model **Tr**, but also of its implementation in **Pr**. This proof of correctness caught some subtle bugs, for example in the definition of $P \,\&\, Q$ there is a clause:

$$P \,\&\, \mathsf{out}\, b\, Q \stackrel{\mathsf{def}}{=} \mathsf{out}\, b\, (P \,\&\, Q) \quad \text{if } P : T \to I$$

This clause was originally not present, which causes subtle buffering errors in the corner case where P's output has completed even though its input has not. For unit testing to catch this bug, a test harness would be required which supports mocking an I/O library to allow testing with incomplete traces. Such mock libraries are difficult to construct, which in turn makes unit testing of I/O-bound programs difficult.

In summary, we have provided an executable library of streaming I/O, together with a mechanically verified proof of its categorical structure. This is the first such library.

4 Related work

Our model is based on monotone functions over traces, and so is strongly related to Kahn dataflow networks [22]. Kahn networks are for streams of type A^ω, and so do not support a notion of stream termination, or concatenation of streams. The main difference between Kahn's model and ours is that we require streams to be consumed in left-to-right order, that is a function of type $f : T \,\&\, U \to V$ must consume all of its T input before consuming any of its U input.

Games models [4, 18] have a similar structure to our model: arenas are (essentially) automata with additional structure, and strategies are (essentially) transducers. Games, however, are designed to have symmetric monoidal structure, rather than braided monoidal structure, since the tensor on types is an interleaving rather than a sequencing of automata. Games models are also compact closed, since they have a dualising action $\cdot^\perp$. This corresponds to the *bidirectional* nature of strategies in a game: a morphism $f : T \to U$ supports input on the right and output on the left, as well as the left-to-right communication allowed by our model.

Session types [17] also provide a bidirectional extension of the types considered here. We can define the first-order output-only sessions in our system as:

$$\mathsf{end} \stackrel{\mathsf{def}}{=} I \quad ![A].T \stackrel{\mathsf{def}}{=} \Sigma(a : A)\, T \quad \oplus\{\ell_i : T_i\} \stackrel{\mathsf{def}}{=} \Sigma(a : \{\ell_1, \ldots, \ell_n\})\, U_a \text{ where } U_{\ell_i} \stackrel{\mathsf{def}}{=} T_i$$

Session types *do* support a notion of terminated session, and so could support a sequential composition operator, but this has not been investigated. Moreover, much of the work on

session types has been in the context of processes rather than functions, with the exception of recent work by Gay and Vasconcelos [11]. There has been no work on categorical session models. Linear logic [14] has symmetric monoidal structure, which has been generalized to the non-commutative case (up to cyclic permutations) by Yetter [30]. Kiselyov's *iteratees* [23] provide a similar model to our transducer process model, although they also support impure iteratees and exceptions. The more fundamental difference between the models is output, which in Millikin's [25] notation is: $\mathsf{Yield}(\mathsf{Chunk}[\vec{b}])[\vec{a}]$. Here, an iteratee is being built with input type A, and output type U; it is outputting $\vec{b}$, and also returning the unconsumed input $\vec{a}$. In our terms, this is a process of type $\langle A\rangle^* \to \langle B\rangle^* \,\&\, \langle A\rangle^*$, that is iteratees are given by the state transformer construction applied to processes. A similar syntactic model for stream processing is provided by Ghani *et al.* [12], and has been generalized to arbitrary coinductive datatypes [13].

5 Future work

Cyclic graphs are modelled categorically as *traced* monoidal categories [21], as discussed by Selinger [27]. Braided traced categories are well-known, but it is not immediately obvious what the right notion of lax braided traced category is. Cyclic graphs would allow modelling of recursive dataflow programs, although there may be a requirement that only contraction maps (with respect to an appropriate metric on traces) can be made cyclic.

Games models and session types are naturally bidirectional, so there exist duals for all types; categorically games form compact closed categories. It is not obvious how to generalize the notion of autonomous category from the braided to the lax braided setting. Compact closed categories are monoidal closed, where $T \Rightarrow U$ is defined to be $T^\perp \otimes U$. This should extend to a higher order sequential model, where $T^\perp \,\&\, U$ is a type for programs which consume all their arguments before producing any results.

There is a natural improvement order on functions, given pointwise by prefix order, for example $\mathsf{swap};\mathsf{swap} \leq \mathsf{id}$. Such an improvement order would make our category a 2-category. These come with natural minimal and maximal elements, for example the the maximal natural transform of type $T \to T$ is the identity function, and the minimal natural transform is a "delay" function that returns ε on any incomplete input, and acts as the identity on complete input. Moreover, **Tr** comes with the right combinators to form a category with finite products, but the equations are only satisfied up to two-cells, for example $\langle f, g\rangle;\pi_2 \leq g$. This may be given an interesting restriction category structure [7] or form a variant of a cartesian bicategory [6].

References

1 The Agda wiki. `http://wiki.portal.chalmers.se/agda/`.

2 The Coq proof assistant. `http://coq.inria.fr/`.

3 The Haskell programming language. `http://www.haskell.org/`.

4 S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163:409–470, 1996.

5 J. Baez and M. Stay. Physics, topology, logic and computation: A Rosetta Stone. In B. Coecke, editor, *New Structures for Physics*, volume 813 of *Lecture Notes in Physics*, chapter 2, pages 95–172. Springer, 2011.

6 A. Carboni and R. F. C. Walters. Cartesian bicategories I. *J. Pure and Applied Algebra*, 49:11–32, 1987.

7 J. R. B. Cockett and S. Lack. Restriction categories I: categories of partial maps. *Theoretical Computer Science*, 270(1-2):223 – 259, 2002.

8 B. Day, E. Panchadcharam, and R. Street. Lax braidings and the lax centre. In L. H. Kauffman, D. E. Radford, and F. J. O. Souza, editors, *Hopf Algebras and Generalizations*, volume 441 of *Contemporary Mathematics*, pages 1–17. AMS, 2007.

9 R. Deline and M. Fähndrich. Typestates for objects. In *Proc. European Conf. Object-Oriented Programming*, pages 465–490. Springer, 2004.

10 F. A. Garside. The braid group and other groups. *Q. J. Mathematics*, 20(1):235–254, 1969.

11 S. Gay and V. T. Vasconcelos. Linear type theory for asynchronous session types. *J. Functional Programming*, 20(1):19–50, 2010.

12 N. Ghani, P. Hancock, and D. Pattinson. Continuous functions on final coalgebras. In *Proc. Coalgebraic Methods in Computer Science*, Electronic Notes in Theoretical Computer Science, 2006.

13 N. Ghani, P. Hancock, and D. Pattinson. Continuous functions on final coalgebras. *Electronic Notes in Theoretical Computer Science*, 249:3–18, 2009. Proc. Mathematical Foundations of Programming Semantics.

14 J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

15 M. Hennessy and G. D. Plotkin. Full abstraction for a simple programming language. In *Proc. Mathematical Foundations of Computer Science*, number 74 in Lecture Notes in Computer Science, pages 108–120. Springer, 1979.

16 C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

17 K. Honda. Types for dyadic interaction. In *Proc. Int. Conf. Concurrency Theory*, number 715 in Lecture Notes in Computer Science, pages 509–523. Springer, 1993.

18 J. M. E. Hyland and C.-H. Ong. On full abstraction for PCF. *Information and Computation*, 163:285–408, 2000.

19 A. S. A. Jeffrey. agda-system-io. `https://github.com/agda/agda-system-io/`, 2010.

20 A. Joyal and R. Street. The geometry of tensor calculus I. *Advances in Mathematics*, 88(1):55–112, 1991.

21 A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Math. Proc. Cambridge Philosophical Society*, 119:447–468, 1996.

22 G. Kahn. The semantics of a simple language for parallel programming. In *Proc. IFIP Congress*, pages 471–475, 1974.

23 O. Kiselyov. Streams and iteratees. `http://okmij.org/ftp/Streams.html`.

24 M. Lothaire. *Applied Combinatorics on Words*, volume 105 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2005.

25 J. Millikin. Understanding iteratees. `http://john-millikin.com/articles/understanding-iteratees/`, 2010.

26 R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

27 P. Selinger. A survey of graphical languages for monoidal categories. In B. Coecke, editor, *New Structures for Physics*, volume 813 of *Lecture Notes in Physics*, chapter 4, pages 289–356. Springer, 2011.

28 D. Sitaram and M. Felleisen. Control delimiters and their hierarchies. *Lisp and Symbolic Computation*, 3(1):67–99, 1990.

29 R. J. van Glabbeek. The linear time – branching time spectrum. In *Proc. Int. Conf. Concurrency Theory*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990.

30 D. Yetter. Quantales and (noncommutative) linear logic. *Journal of Symbolic Logic*, 55(1):41–64, 1990.

The Church Synthesis Problem with Metric*

Mark Jenkins[1], Joël Ouaknine[1], Alexander Rabinovich[2], and James Worrell[1]

1 **Department of Computer Science, University of Oxford, Oxford, UK**
{marj,joel,jbw}@cs.ox.ac.uk
2 **School of Computer Science, Tel Aviv University, Tel Aviv, Israel**
rabinoa@post.tau.ac.il

Abstract

Church's Problem asks for the construction of a procedure which, given a logical specification $\varphi(I, O)$ between input strings I and output strings O, determines whether there exists an operator F that implements the specification in the sense that $\varphi(I, F(I))$ holds for all inputs I. Büchi and Landweber gave a procedure to solve Church's problem for MSO specifications and operators computable by finite-state automata.

We consider extensions of Church's problem in two orthogonal directions: (i) we address the problem in a more general logical setting, where not only the specifications but also the solutions are presented in a logical system; (ii) we consider not only the canonical discrete time domain of the natural numbers, but also the continuous domain of reals.

We show that for every fixed bounded length interval of the reals, Church's problem is decidable when specifications and implementations are described in the monadic second-order logics over the reals with order and the +1 function.

1998 ACM Subject Classification F.4.1 Mathematical Logic; F.4.3 Formal Languages

Keywords and phrases Church's Problem, monadic logic, games, uniformization

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.307

1 Introduction

Church's *synthesis problem* [3] is to automatically construct an implementation of a specification relating the inputs and outputs of a state-based system. The specification is assumed to be an $\mathsf{MSO}(<)$-formula $S(I, O)$, which determines a binary relation between input strings I and output strings O. An implementation is a function (or operator) P from strings to strings that *uniformizes* S in the sense that $S(I, P(I))$ holds for all inputs I. Church required that P be computable by a finite-state machine that at every moment $t \in \mathbb{N}$ reads an input symbol $I(t)$ and produces an output symbol $O(t)$. Hence, the output $O(t)$ produced at t depends only on input symbols $I(0), I(1), \ldots, I(t)$ received before t, that is, P should be a *causal* operator. Another property of interest is that the machine computing P be finite-state. In the light of Büchi's proof [1] of the expressive equivalence of $\mathsf{MSO}(<)$ and finite automata, P is finite-state if and only if it is $\mathsf{MSO}(<)$-definable.

Church's synthesis problem can therefore be stated formally as follows.

* The third author was partially supported by the ESF GAMES programme and by the EPSRC.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 307–321

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

Church Synthesis Problem

Input: an $\mathsf{MSO}(<)$ formula $\varphi(X, Y)$.
Task: Check whether there is a causal operator F such that
$\langle \mathbb{N}, < \rangle \models \forall X \varphi(X, F(X))$ and if so, construct this operator.

This problem, which is more general than the satisfiability problem for MSO over $\langle \mathbb{N}, < \rangle$, was shown decidable in a landmark paper of Büchi and Landweber [2]. Their main theorem is stated as follows:

▶ **Theorem 1** (Büchi and Landweber). *Given an* $\mathsf{MSO}(<)$ *formula* $\varphi(\overline{X}, \overline{Y})$ *one can decide whether there is a causal operator that uniformizes* φ*. If such an operator exists then it can be represented by a finite-state automaton which can be computed from* φ*.*

Note that this theorem guarantees that whenever φ has a uniformizer then it has a uniformizer that is computable by a finite-state automaton (equivalently, definable in $\mathsf{MSO}(<)$).

In the continuous-time setting, one can naturally consider the synthesis problem over the non-negative reals rather than the naturals. Here we think of a specification as a relation between *signals* rather than words. As specification language one again takes $\mathsf{MSO}(<)$, which has a natural interpretation over the non-negative reals. As implementations one again takes $\mathsf{MSO}(<)$-definable causal operators.

Shelah [13] proved that $\mathsf{MSO}(<)$ is undecidable over the reals if we allow quantification over arbitrary predicates. In Computer Science however, it is natural to restrict to finitely variable predicates, that is, predicates whose characteristic function has finitely many discontinuities in any bounded interval. Under the finite-variability interpretation, MSO is decidable over the reals and there are automata that have the same expressive power [10].

However, the full extension of the Büchi and Landweber theorem fails over the nonnegative reals, even under the finite-variability assumption. For example, the formula that says that Y has at least two points of discontinuity can be uniformized by a causal operator, but not by an $\mathsf{MSO}(<)$*-definable causal operator* (see Example 7 for details).

Nevertheless, we are able to show the following result:

▶ **Theorem 2.** *Given a* $\mathsf{MSO}(<)$ *formula* $\varphi(\overline{X}, \overline{Y})$ *one can decide whether there is an* $\mathsf{MSO}(<)$*-definable causal operator that uniformizes* φ *over* $\langle \mathbb{R}_{\geq 0}, < \rangle$*. If so, the algorithm computes a formula that defines such an operator.*

In the continuous setting a deficiency of $\mathsf{MSO}(<)$ is that it cannot express metric properties such as "the distance between two points is one". Thus we consider specifications expressed in $\mathsf{MSO}(<, +1)$, which extends $\mathsf{MSO}(<)$ with the $+1$ function.

Unfortunately, even with the finitely-variable interpretation, the satisfiability problem over the non-negative reals is undecidable for $\mathsf{MSO}(<, +1)$ [5]. However in [9] we proved that $\mathsf{MSO}(<, +1)$ is decidable for every fixed bounded-length intervals of the reals. The main result of our paper is that Church's synthesis problem is also decidable for $\mathsf{MSO}(<, +1)$ for every fixed bounded-length interval of the reals. Specifically,

▶ **Theorem 3.** *Given an* $\mathsf{MSO}(<, +1)$ *formula* $\varphi(X, Y)$ *and* $N \in \mathbb{N}$*, one can decide whether there is an* $\mathsf{MSO}(<, +1)$*-definable causal operator that uniformizes* φ *over the the interval* $[0, N)$*. If such an operator exists, the algorithm computes a formula that represents the operator.*

In order to prove Theorem 3 we need to consider the Church synthesis problem with *parameters*—additional predicates that the specification may reference but which do not have

to be considered in a causal way by the implementation. This problem was considered in [11] for MSO$(<)$ over $\langle \mathbb{N}, < \rangle$. Here we extend the Church synthesis problem with parameters to the non-negative reals.

Finally, we show that the synthesis problem over bounded intervals of reals is non-elementary, even for specifications expressed in fragments of MSO$(<, +1)$ with an elementary satisfiability problem.

2 Monadic Second-Order Logic

We consider monadic second-order logic MSO$(<, +1)$ over a signature consisting of the binary relations $<$ and $+1$ and a countable family of monadic predicate names $P_0, P_1, \ldots$. The vocabulary of MSO$(<, +1)$ also includes *first-order* variables $t_0, t_1, \ldots$ and monadic *second-order* variables $X_1, X_2, \ldots$. Atomic formulas are of the form $X(t)$, $P(t)$, $t_1 < t_2$, $+1(t_1, t_2)$ or $t_1 = t_2$. Well-formed formulas are obtained from atomic formulas using Boolean connectives, the first-order quantifiers $\exists t$ and $\forall t$, and the second-order quantifiers $\exists X$ and $\forall X$. We denote by MSO$(<)$ the sub-language consisting of all formulas that do not mention the $+1$ relation.

We are interested in structures of the form $\mathcal{M} = \langle A, <, +1, \mathbf{P}_1, \ldots, \mathbf{P}_m \rangle$ where A is an interval of non-negative reals with the usual order, $+1(x, y)$ holds if and only if $y = x + 1$, and $\mathbf{P}_1, \ldots, \mathbf{P}_m$ are subsets of A that interpret the monadic predicate names $P_1, \ldots, P_m$. (Generally we use boldface to denote interpretations of predicate names.) We omit the standard definition of what it means for a structure to satisfy a sentence. A formula $\varphi(P_1, \ldots, P_m, X_1, \ldots, X_n)$ with free second-order variables among $X_1, \ldots, X_n$ is interpreted in a structure $\langle \mathcal{M}, \mathbf{X}_1, \ldots, \mathbf{X}_n \rangle$ obtained by expanding $\mathcal{M}$ with interpretations of $X_1, \ldots, X_n$.

We say that a subset $\mathbf{P} \subseteq \mathbb{R}_{\geq 0}$ is *finitely variable* if its characteristic function has finitely many discontinuities in any bounded sub-interval of $\mathbb{R}_{\geq 0}$. Likewise we say that $\mathbf{P}$ is *right-continuous* if its characteristic function is right continuous. In our semantics for MSO$(<, +1)$ we restrict interpretations of monadic predicate names and variables to be finitely variable and right-continuous. The finite variability restriction is essential: it is known that allowing unrestricted second-order quantification leads to an undecidable satisfiability problem [13]. On the other hand, the assumption of right-continuity is only for simplicity of presentation. In case the domain A is unbounded we also make the simplifying assumption that at least one of the predicates in any structure $\langle A, <, \overline{\mathbf{P}} \rangle$ is not eventually constant.

We also consider discrete structures for MSO$(<)$ of the form $\langle A, <, \mathbf{P}_1, \ldots, \mathbf{P}_m \rangle$, where A is an initial segment of the natural numbers with the usual order and $\mathbf{P}_1, \ldots, \mathbf{P}_m$ are subsets of A. In line with our assumption for structures over the reals we assume that if $A = \mathbb{N}$ then in any structure $\langle A, <, \overline{\mathbf{P}} \rangle$, at least one of the predicates is not eventually constant.

Let $\Sigma_P = \{0, 1\}^m$ be a finite alphabet. A structure $\langle A, <, \mathbf{P}_1, \ldots, \mathbf{P}_m \rangle$ corresponds to a function $f : A \to \Sigma_P$, where $f(t)_i = 1$ if $t \in \mathbf{P}_i$ and $f(t)_i = 0$ otherwise. If $A \subseteq \mathbb{R}_{\geq 0}$ then f is called a *signal* and if $A \subseteq \mathbb{N}$ then f is a (finite or infinite) word. By assumption, a signal is finitely variable, right-continuous and if its domain is unbounded it is not eventually constant. We denote by Σ^ω the set of all infinite words over an alphabet Σ and by $Sig(\Sigma)$ the set of all signals over Σ with domain $\mathbb{R}_{\geq 0}$. An MSO$(<)$-sentence $\varphi(\overline{P})$ that mentions predicate names $P_1, \ldots, P_m$ respectively defines a word language $L_{\mathbb{N}}(\varphi) \stackrel{\text{def}}{=} \{w \in (\Sigma_P)^\omega : w \models \varphi\}$ and a signal language $L_{\mathbb{R}}(\varphi) \stackrel{\text{def}}{=} \{f \in Sig(\Sigma_P) : f \models \varphi\}$.

3 Transforming Between Words and Signals

In this paper we answer questions about MSO over the non-negative reals under the finite variability and right-continuous interpretation by reduction to questions about MSO over the naturals. This section presents the foundations of this reduction—semantic translations between signal languages and word languages and corresponding syntactic translations on MSO(<) formulas. We concentrate here on signals with domain $\mathbb{R}_{\geq 0}$ and on infinite words, though the ideas easily apply to signals with bounded domain and to finite words.

Let $f : \mathbb{R}_{\geq 0} \to \Sigma$ be a signal over alphabet Σ. Recall that by assumption f has a countably infinite and unbounded set of discontinuities. Define a *sampling sequence* for f to be an unbounded strictly increasing sequence of reals $0 = \tau_0 < \tau_1 < \tau_2 < \ldots$ that includes all discontinuities of f. Given a sampling sequence τ we define the word $W_\tau(f) \in \Sigma^\omega$ by $W_\tau(f) = f(\tau_0)f(\tau_1)f(\tau_2)\ldots$. Given a language $L \subseteq \mathit{Sig}(\Sigma)$ we define the corresponding word language $L^\dagger \subseteq \Sigma^\omega$ to comprise all words $W_\tau(f)$ where $f \in L$ and τ is a sampling sequence for f.

Motivated by the translation above we define the relation $\sim$ of *stutter equivalence* on Σ^ω to be the least equivalence relation such that

$$a_0a_1 \ldots a_{k-1}a_ka_{k+1} \ldots \sim a_0a_1 \ldots a_{k-1}a_ka_ka_{k+1} \ldots$$

for any k. A word language $L \subseteq \Sigma^\omega$ is *stutter-closed* if it saturates $\sim$. It is straightforward that if L is a signal language then the corresponding word language $L^\dagger$ is stutter-closed.

Define the *stutter-closure* of a language $L \subseteq \Sigma^\omega$ to be the smallest stutter-closed language L' that contains L. It is straightforward that L' is ω-regular if L is ω-regular.

Given a word $w = w_0w_1w_2 \ldots \in \Sigma^\omega$ and an unbounded strictly increasing sequence of reals $0 = \tau_0 < \tau_1 < \tau_2 < \ldots$, define the signal $S_\tau(w) : \mathbb{R}_{\geq 0} \to \Sigma$ by $S_\tau(t) = w_k$ for the unique interval $[\tau_k, \tau_{k+1})$ containing t. Given a word language $L \subseteq \Sigma^\omega$, the corresponding signal language $L^\star$ comprises all signals $S_\tau(w)$ for some $w \in L$ and unbounded sequence of reals τ.

Define the relation $\sim$ of *stretching equivalence* on the set $\mathit{Sig}(\Sigma)$ of signals over alphabet Σ by $f \sim g$ iff $f = g \circ \rho$ for some order isomorphism $\rho : \mathrm{dom}(f) \to \mathrm{dom}(g)$. A signal language L is *speed-independent* if it saturates $\sim$. It is straightforward that $L_{\mathbb{R}}(\varphi)$ is speed-independent for any MSO(<)-formula φ. It is also clear that $L^\star$ is speed-independent for any word language $L \subseteq \Sigma^\omega$.

The operators $(-)^\dagger$ and $(-)^\star$ define a bijection between stutter-closed word languages and speed-independent signal languages:

▶ **Proposition 4.** *If $L \subseteq \Sigma^\omega$ is stutter-closed then $L^{\star\dagger} = L$. If $L \subseteq \mathit{Sig}(\Sigma)$ is speed independent then $L^{\dagger\star} = L$.*

Next we recall from [10] syntactic analogs of the language operators $(-)^\dagger$ and $(-)^\star$. The following results show that the language operators $(-)^\dagger$ and $(-)^\star$ preserve MSO-definability. We briefly justify these constructions and refer the reader to [10] for full details.

▶ **Proposition 5.** *Given an MSO(<)-sentence $\varphi(\overline{P})$ we can compute another MSO(<)-sentence $\varphi^\dagger(\overline{P})$ such that $L_{\mathbb{R}}(\varphi)^\dagger = L_{\mathbb{N}}(\varphi^\dagger)$.*

Proof. (Sketch.) In order to define $\varphi^\dagger$ we first rewrite φ by replacing first-order variables with second-order variables. Intuitively we represent an element $t \in \mathbb{R}_{\geq 0}$ by a set with left endpoint t. (Recall that the restriction to finitely variable right-continuous signals means any set interpreting a second-order variable has a least element.) To this end we introduce the following new atomic formulas: $Ls(X, Y)$, which is true if the left endpoint of X is less

than the left endpoint of Y; $Eq(X,Y)$, which is true if the left endpoint of X equals the left endpoint of Y; $In(X,Y)$, which is true if the left endpoint of X is an element of Y. These atoms are clearly MSO(<)-definable, moreover it is straightforward to construct a formula equivalent to φ using only these atoms, Boolean connectives and second-order quantification.

Having performed this translation, we proceed to define $\varphi^\dagger$ by structural induction on φ. The new atomic formulas $Ls(X,Y)$, $Eq(X,Y)$ and $In(X,Y)$ go unchanged, as do negation and conjunction. The only non-trivial element of the transformation concerns second-order quantification. If $\varphi \stackrel{\text{def}}{=} \exists X\psi$ we let $\varphi^\dagger$ be the formula that defines the stutter closure of the language $L_{\mathbb{N}}(\exists X\psi^\dagger)$. (We have already observed above that this language is definable.) After this process it is trivial to replace occurrences of $Ls(X,Y)$, $Eq(X,Y)$ and $In(X,Y)$ with their MSO(<)-equivalents. ◀

▶ **Proposition 6.** *Given an* MSO(<)*-sentence* $\varphi(\overline{P})$ *we can compute another* MSO(<)*-sentence* $\varphi^\star(\overline{P})$ *such that* $L_{\mathbb{R}}(\varphi^\star) = L_{\mathbb{N}}(\varphi)^\star$.

Proof. (Sketch.) It is straightforward to write an MSO(<)-formula $Sample(\overline{P}, D)$ that is satisfied when each discontinuity of the characteristic signal of $\overline{P}$ is also a discontinuity of the characteristic signal of D. Given $\varphi(\overline{P})$ let $\varphi'(\overline{P}, D)$ be obtained by relativizing all first-order quantification in φ to the set of discontinuities in D. We now define

$$\varphi^\star := \exists D(Sample(\overline{P}, D) \wedge \varphi'(\overline{P}, D))\,. \qquad ◀$$

4 Church's Problem with Parameters

Recall that a binary relation R is *uniformized* by a partial function f if $f \subseteq R$ and $\mathrm{dom}(f) = \mathrm{dom}(R)$. In this paper we are interested in uniformizing MSO-definable relations by MSO-definable functions. The problem of uniformizing MSO-definable relations on the structure $\langle \mathbb{N}, < \rangle$ was first studied over fifty years ago by Church [3], motivated by the problem of synthesizing circuits from relational input-output specifications. Later Rabinovich [11] and Hänsch, Slaats and Thomas [4] considered the problem of uniformization over *labelled chains* $\langle \mathbb{N}, <, \overline{\mathbf{P}} \rangle$.

Our eventual goal is to study uniformization of MSO-definable relations over the structure $\langle A, <, +1 \rangle$ for A a bounded interval of reals. We delay a treatment of the $+1$ relation until the following section. Here we lay the groundwork by considering uniformization of labelled chains $\langle \mathbb{R}_{\geq 0}, <, \overline{\mathbf{P}} \rangle$. This extends the treatment of the labelled case from [4, 11] to dense orders.

4.1 The Uniformization Problem

Consider a second-order language over a signature including the binary relation symbol $<$ and monadic predicate names $P_1, \ldots, P_m$. Let $\mathcal{M} = \langle A, <, \mathbf{P}_1, \ldots, \mathbf{P}_m \rangle$ be a labelled chain. We say that an MSO-formula $\psi(\overline{P}, \overline{X}, \overline{Y})$ uniformizes an MSO-formula $\varphi(\overline{P}, \overline{X}, \overline{Y})$ over $\mathcal{M}$ if $\mathcal{M}$ satisfies the following sentences:

1. $\forall \overline{X} \forall \overline{Y} \left(\psi(\overline{P}, \overline{X}, \overline{Y}) \rightarrow \varphi(\overline{P}, \overline{X}, \overline{Y}) \right)$
2. $\forall \overline{X} \exists^{=1} \overline{Y} \psi(\overline{P}, \overline{X}, \overline{Y})$

We say that $\psi(\overline{P}, \overline{X}, \overline{Y})$ uniformizes $\varphi(\overline{P}, \overline{X}, \overline{Y})$ over a class of chains $\mathcal{C}$ if $\psi(\overline{P}, \overline{X}, \overline{Y})$ uniformizes $\varphi(\overline{P}, \overline{X}, \overline{Y})$ over each individual chain in $\mathcal{C}$. Notice that the above conditions can only hold if $\varphi(\overline{P}, \overline{X}, \overline{Y})$ is a total relation, however there is no loss of generality in considering only uniformization for total relations.

We say that a formula $\psi(\overline{P}, \overline{X}, \overline{Y})$ satisfying 1 above is *faithful* to φ and a formula satisfying 2 above is *functional*. We furthermore say that ψ is *causal* if the following sentence holds in $\mathcal{M}$:

3. $\forall \overline{X}\, \overline{Y}\, \overline{U}\, \overline{V}\, \forall t\ \left[\psi(\overline{P}, \overline{X}, \overline{Y}) \wedge \psi(\overline{P}, \overline{U}, \overline{V}) \wedge \left(\forall s \leq t\, (\overline{X}(s) = \overline{U}(s))\right) \Rightarrow \overline{Y}(t) = \overline{V}(t)\right]$

Intuitively a function is causal if its output at any time only depends on its input in the past—a reasonable assumption for any realizable function.

Roughly speaking, the uniformization problem is to determine whether a given formula $\varphi(\overline{P}, \overline{X}, \overline{Y})$ has a uniformizer over a given structure, or class of structures, and if so to compute such a uniformizer. We are interested here in uniformizers which are definable by an MSO formula and this proves an important restriction over structures with real-valued domains. The following example illustrates a case where one can easily think of a uniformizer for φ, but no such uniformizer can be definable in MSO.

▶ **Example 7.** There is a formula $\varphi(X, Y)$ (even without parameters) such that there is a causal operator which uniformizes φ over the reals however, no MSO-definable causal operator uniformizes it.

Proof. Let $\varphi(X, Y)$ be an MSO($<$)-formula which says that Y has at least two points of discontinuity. It is clear that the operator F which ignores its input and sets $F(X)(t) = 1$ if and only if $t \in ([0, 1) \cup [2, 3))$ uniformizes this formula, however we prove below there is no MSO-definable uniformizer $\psi(X, Y)$.

Let $\psi(X, Y)$ be an MSO($<$)-formula that defines a functional operator. Interpret X by the constant false signal $\mathbf{X}$ and let $\mathbf{Y}$ be the unique interpretation of Y such that $\psi(\mathbf{X}, \mathbf{Y})$ holds. For any order isomorphism $\rho : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, since every MSO($<$)-formula is speed-independent, $\psi(\mathbf{X} \circ \rho, \mathbf{Y} \circ \rho)$ also holds. But $\mathbf{X} \circ \rho = \mathbf{X}$; as ψ is functional we must also have $\mathbf{Y} \circ \rho = \mathbf{Y}$. It is easy to see that this entails that $\mathbf{Y}$ be constant, contrary to the requirement that it have two discontinuities. ◀

Motivated by failures such as this, we seek to compute the set of parameter values for which there exists a definable uniformizer along with a single formula which defines a uniformizer for all such parameter values. We formally state the main result of this section as follows:

▶ **Theorem 2.** *Given an* MSO($<$)*-formula* $\varphi(\overline{P}, \overline{X}, \overline{Y})$ *one can compute a sentence* $\theta(\overline{P})$ *and formula* $\psi(\overline{P}, \overline{X}, \overline{Y})$ *such that for every structure* $\mathcal{M} = \langle \mathbb{R}_{\geq 0}, <, \overline{\mathbf{P}} \rangle$*,* φ *has a causal uniformizer over* $\mathcal{M}$ *if and only if* $\mathcal{M} \models \theta$ *and in this case* ψ *is such a causal uniformizer.*

In Theorem 2 we call φ the *winning condition*, ψ the *uniformizer* and θ the *domain formula*. We call the predicate names $\overline{P}$ *parameters* and $\overline{X}, \overline{Y}$ *variables*.

We sketch a proof of Theorem 2 in Sections 4.2 and 4.3.

4.2 From Signals to Words

Let L_1 be a speed-independent language of signals over alphabet $\Sigma_P = \{0, 1\}^m$ and let $L_2 = (L_1)^\dagger$ be the corresponding stutter-closed language of words. We identify a signal f with the corresponding structure $\langle \mathbb{R}_{\geq 0}, <, \mathbf{P}_1, \ldots, \mathbf{P}_m \rangle$, where f is the characteristic function of $\overline{\mathbf{P}}$; we similarly identify a word w with the corresponding structure $\langle \mathbb{N}, <, \overline{\mathbf{P}} \rangle$. We further identify a language L with the class of stuctures that correspond to the elements of L.

We reduce the problem of computing a uniformizer of an MSO($<$)-formula $\varphi(\overline{P}, \overline{X}, \overline{Y})$ over the class of signals L_1 to the problem of computing a uniformizer for the corresponding formula $\varphi^\dagger(\overline{P}, \overline{X}, \overline{Y})$ over the class of words L_2. Superficially these two problems are quite

different since L_1 is a class of dense orders and L_2 a class of discrete orders. The key fact that makes this reduction work is that for each signal $f \in L_1$ we include in L_2 the whole stutter-closed class of words representing f. Given this, the reduction is simply stated:

▶ **Theorem 8.** *If $\psi(\overline{P}, \overline{X}, \overline{Y})$ is a causal uniformizer for $\varphi(\overline{P}, \overline{X}, \overline{Y})$ over L_1 then $\psi^\dagger(\overline{P}, \overline{X}, \overline{Y})$ is a stutter-closed causal uniformizer for $\varphi^\dagger(\overline{P}, \overline{X}, \overline{Y})$ over L_2. Conversely if $\psi(\overline{P}, \overline{X}, \overline{Y})$ is a stutter-closed causal uniformizer for $\varphi^\dagger(\overline{P}, \overline{X}, \overline{Y})$ over L_2, then $\psi^\star(\overline{P}, \overline{X}, \overline{Y})$ is a causal uniformizer for $\varphi(\overline{P}, \overline{X}, \overline{Y})$ over L_1.*

The proof of Theorem 8 relies on the relations between speed-independent signal languages and stutter-closed word languages developed in Section 3.

We will use Theorem 8 to reduce the problem of uniformizing classes of signals, considered in Theorem 2, to the problem of computing stutter-closed uniformizers of stutter-closed formulas over words. We therefore undertake to prove the following Theorem.

▶ **Theorem 9.** *Given a stutter-closed $\mathsf{MSO}(<)$-formula $\varphi(\overline{P}, \overline{X}, \overline{Y})$ one can compute a stutter-closed sentence $\theta(\overline{P})$ and stutter-closed formula $\psi(\overline{P}, \overline{X}, \overline{Y})$ such that for any stutter-closed language of words $L \subseteq (\Sigma_P)^\omega$, φ has a causal uniformizer over L if and only if $L \models \theta$ and in this case ψ is such a causal uniformizer.*

Considering φ over signals, we observe that the word language defined by $\varphi^\dagger$ is always stutter-closed. We therefore first apply Theorem 9 then Theorem 8 to $\varphi^\dagger$ to derive Theorem 2.

4.3 Stutter-Closed Uniformizers

Say that a formula $\psi(\overline{P}, \overline{X}, \overline{Y})$ defines a *stutter-preserving* relation on $\mathcal{M} = \langle \mathbb{N}, <, \overline{\mathbf{P}} \rangle$ if $\mathcal{M}$ satisfies $\forall \overline{X}, \overline{Y}(\psi(\overline{P}, \overline{X}, \overline{Y}) \to \mathit{StutPres}(\overline{P}, \overline{X}, \overline{Y}))$, where

$$\mathit{StutPres}(\overline{P}, \overline{X}, \overline{Y}) \stackrel{\text{def}}{=} \forall n(\overline{X}(n) = \overline{X}(n+1) \wedge \overline{P}(n) = \overline{P}(n+1) \to \overline{Y}(n) = \overline{Y}(n+1)) \quad (1)$$

In other words, in the function defined by ψ the output $\overline{Y}$ can only change when either the input $\overline{X}$ or parameters $\overline{P}$ change.

The following is straightforward.

▶ **Proposition 10.** *Let $L \subseteq (\Sigma_P)^\omega$ be a stutter-closed language of words. If $\psi(\overline{P}, \overline{X}, \overline{Y})$ is functional over L and stutter-closed then it is also stutter-preserving over L.*

Say that an ω-word $u = u_0 u_1 \ldots$ is *stutter-free* if $u_i \neq u_{i+1}$ for all i. Recall that we assume that for any structure $\langle \mathbb{N}, <, \overline{\mathbf{P}} \rangle$ one of the predicates $\mathbf{P}_i$ is not eventually constant. This means that the characteristic ω-word of the structure is stutter equivalent to a unique stutter-free word.

Proof of Theorem 9. We define a game based on φ such that the uniformizer ψ defines a winning strategy in this game. Our proof is based on the construction of Hänsch, Slaats and Thomas [4] but requires non-trivial modification to handle various issues related to stuttering.

Step 1: *definition of game arena $\mathcal{G}$.* Define a formula $\varphi'(\overline{P}, \overline{X}, \overline{Y})$ by

$$\varphi'(\overline{P}, \overline{X}, \overline{Y}) \stackrel{\text{def}}{=} (\varphi(\overline{P}, \overline{X}, \overline{Y}) \wedge \mathit{StutPres}(\overline{P}, \overline{X}, \overline{Y})) \vee \mathit{EvConst}(\overline{P}), \quad (2)$$

where *StutPres* is the formula in (1) expressing stutter preservation and $\mathit{EvConst}(\overline{P})$ expresses that $\overline{P}$ is eventually constant. The inclusion of *StutPres* is justified by the observation in Proposition 10 that a stuttering-closed uniformizer is stutter-preserving. The inclusion of

EvConst$(\overline{P})$ is connected with our semantic assumption that the characteristic words of structures over $\mathbb{N}$ are not eventually constant.

The formula φ' mentions predicate names $\overline{P} = (P_1, \ldots, P_m)$ and free variables $\overline{X} = (X_1, \ldots, X_n)$ and $\overline{Y} = (Y_1, \ldots, Y_\ell)$. Then interpretations for φ' over domain $\mathbb{N}$ are ω-words over the alphabet $\{0,1\}^m \times \{0,1\}^n \times \{0,1\}^\ell$. The first step is to construct a deterministic parity automaton $\mathcal{A}$ over this alphabet that accepts precisely those words that satisfy φ'. We transform this automaton into a parity game arena $\mathcal{G}$ by separating each transition $s \xrightarrow{(p,x,y)} t$ of $\mathcal{A}$ into a pair of transitions $s \longrightarrow (s,p,x)$ and $(s,p,x) \longrightarrow t$ controlled by Player 1 and Player 2 respectively. The priorities of the states in $\mathcal{G}$ are inherited from $\mathcal{A}$.

Step 2: *definition of parity game* $\mathcal{G}^\pi$. Next, given a stutter-free ω-word π over alphabet $\{0,1\}^m$, representing an interpretation of the parameters, we transform the arena $\mathcal{G}$ into an infinite-state parity game denoted $\mathcal{G}^\pi$. This game is stratified into finite levels—one level for each letter of π, the states at each level being a copy of those of $\mathcal{G}$. Multiple rounds of the game can be played at each level, with Player 1 controlling passage from one level to the next.

The states of $\mathcal{G}^\pi$ are pairs consisting of a state of $\mathcal{G}$ and a *level number* i. For each Player-1 edge $s \longrightarrow (s,p,x)$ in $\mathcal{G}$ and index $i \in \mathbb{N}$ we include a Player-1 edge $(s)_i \longrightarrow (s,p,x)_i$ if $p = \pi_i$ and an edge $(s)_i \longrightarrow (s,p,x)_{i+1}$ if $p = \pi_{i+1}$. For each Player-2 edge $(s,p,x) \longrightarrow t$ in $\mathcal{G}$ and index $i \in \mathbb{N}$ we include an edge $(s,p,x)_i \longrightarrow (t)_i$ in $\mathcal{G}^\pi$. Finally we add a new initial Player-1 state $(\hat{s})_0$ to $\mathcal{G}^\pi$, where s is the initial state of $\mathcal{G}$. From this state there is an edge to a state $(s,p,x)_0$ if $p = \pi_0$.

Note that due to the disjunct *EvConst*$(\overline{P})$ in (2) Player 1 cannot win $\mathcal{G}^\pi$ by choosing never to leave a given level.

The key property of the game $\mathcal{G}^\pi$, which depends on the fact that $\mathcal{A}$ is stutter-closed, is as follows:

> *Player 2 wins* $\mathcal{G}^\pi$ *if and only if* $\varphi(\overline{P}, \overline{X}, \overline{Y})$ *has a causal uniformizer over the class of all infinite words* π' *that are stutter equivalent to* π.

The easier direction in the proof of the above claim is the right-to-left implication: one can easily show that a causal uniformizer for $\varphi(\overline{P}, \overline{X}, \overline{Y})$ over the stutter equivalence class of π yields a winning strategy in $\mathcal{G}^\pi$. Below we concentrate on the left-to-right implication.

Step 3: *coding and testing strategies.* As $\mathcal{G}^\pi$ is a parity game it is determined and has memoryless winning strategies. To compute winning strategies we first divide the set of game states into levels S_i which contain those nodes annotated with level number i. We can encode the possible levels by a finite alphabet Σ and thus represent the game as an ω-word $\sigma \in \Sigma^\omega$ in which σ_i represents level S_i. Note that σ can be produced as the output of a transducer $\mathcal{S}$ on input π.

A memoryless strategy for Player 2 in $\mathcal{G}^\pi$ maps each node $(s,p,x)_i$ to a node $(t)_i$ and can be represented by an word γ over a finite alphabet Γ whose letters encode the finite sub-strategy for each level i. We build a deterministic parity automaton $\mathcal{T}$ that takes as input pairs of words π and γ and accepts if and only if the strategy γ is winning in $\mathcal{G}^\pi$. The automaton $\mathcal{T}$ incorporates the transducer $\mathcal{S}$ to transform the input word π into a word σ representing the game $\mathcal{G}^\pi$ in the manner described above. For each level i of $\mathcal{G}^\pi$ and level-i Player-2 strategy γ_i, automaton $\mathcal{T}$ computes the finite set of all possibilities over Player-1 moves for the first state, last state and lowest-priority intermediate state of the level-i segment of of a play of $\mathcal{G}^\pi$.

The strategy tester automaton $\mathcal{T}$ is equivalent to an $\mathsf{MSO}(<)$-formula $\chi(\overline{P}, \overline{S})$, where $\overline{S}$ encodes letters of the strategy alphabet Γ. Note that χ is only satisfied by stutter-free

interpretations of $\overline{P}$.

Step 4: *selecting a winning strategy.* The formula $\chi(\overline{P},\overline{S})$ allows us to detect when $\overline{S}$ encodes a winning strategy in $\mathcal{G}^\pi$. A key issue here is that there may be more than one winning strategy for a given set of parameters π: a uniformizer corresponds to a particular winning strategy.

We can compute an $\mathsf{MSO}(<)$-formula that picks a winning strategy using a result of Lifsches and Shelah [8] on the computability of selectors in $\mathsf{MSO}(<)$. We say that a formula $\alpha(\overline{P},\overline{S})$ is a *selector* for a formula $\beta(\overline{P},\overline{S})$ over a structure $\mathcal{M}=\langle\mathbb{N},<,\overline{\mathbf{P}}\rangle$ if:

1. $\mathcal{M}\models\exists^{\leq 1}\overline{S}\,\alpha(\overline{P},\overline{S})$;
2. $\mathcal{M}\models\forall\overline{S}(\alpha(\overline{P},\overline{S})\rightarrow\beta(\overline{P},\overline{S}))$;
3. $\mathcal{M}\models(\exists\overline{S}\beta(\overline{P},\overline{S}))\rightarrow(\exists\overline{S}\alpha(\overline{P},\overline{S}))$.

▶ **Lemma 11** (Selector Lemma [12]). *There is an algorithm that for every formula $\beta(\overline{P},\overline{S})$ constructs a formula $\alpha(\overline{P},\overline{S})$ such that α is a selector for β over all structures $\mathcal{M}=\langle\mathbb{N},<,\overline{\mathbf{P}}\rangle$.*

Applying Lemma 11 we can compute a selector $\chi'(\overline{P},\overline{S})$ for $\chi(\overline{P},\overline{S})$. Then $\chi'(\overline{P},\overline{S})$ is satisfied when $\overline{P}$ is interpreted by a stutter-free word π and $\overline{S}$ is interpreted by a word representing a winning strategy γ in $\mathcal{G}^\pi$.

Step 5: *definition of $\theta(\overline{P})$ and $\psi(\overline{P},\overline{X},\overline{Y})$.*

Similar to the definition of the strategy tester automaton we can compute a formula $\mathit{Strat}(\overline{P},\overline{X},\overline{Y},\overline{S})$ that is true precisely when in $\mathcal{G}^\pi$, for π the unique stutter-free word equivalent to $\overline{P}$, the sequence of moves $\overline{X}$ by Player 1 generates the sequence of responses $\overline{Y}$ by Player 2 given that his strategy on the k-th round is $\overline{S}(k)$.

Note that $\mathit{Strat}(\overline{P},\overline{X},\overline{Y},\overline{S})$ defines a stutter-closed language. Closure under removing stutters follows from the fact that $\overline{S}$ encodes a memoryless strategy and (it can be assumed without loss of generality that) $\mathcal{A}$ doesn't change state when its input stutters. Closure under adding stutters follows similarly using in addition the fact that $\overline{S}$ encodes a stutter-preserving strategy.

We also define $\chi''(\overline{P},\overline{S})$ to be the stutter-closure of the strategy selection operator $\chi'(\overline{P},\overline{S})$. To define χ'' consider positions where $\overline{P}$ changes and state that χ' holds over this set of positions (just relativization) and that $\overline{S}$ does not change between changes of $\overline{P}$.

Finally we are able to define

$$\begin{aligned}\theta(\overline{P}) &\stackrel{\text{def}}{=} \exists\overline{S}\chi''(\overline{P},\overline{S})\\ \psi(\overline{P},\overline{X},\overline{Y}) &\stackrel{\text{def}}{=} \exists\overline{S}\left(\chi''(\overline{P},\overline{S})\wedge\mathit{Strat}(\overline{P},\overline{X},\overline{Y},\overline{S})\right).\end{aligned}$$

Then both θ and ψ are stutter-closed since both χ'' and *Strat* are stutter-closed.

By construction, $\theta(\overline{\mathbf{P}})$ holds if and only if there exists $\overline{\mathbf{S}}$ that encodes a winning strategy for Player 2 in $\mathcal{G}^\pi$, where π the unique stutter-free word equivalent to the characteristic ω-word $u_{\overline{\mathbf{P}}}$. Since *Strat* encodes plays of this game that follow this winning strategy, ψ uniformizes φ.

This concludes the proof of Theorem 9. ◀

5 Uniformizing Metric Formulas

In this section we show decidability of the uniformization problem for $\mathsf{MSO}(<,+1)$ over bounded real time domains.

Note that as an immediate corollary of Theorem 2, we can establish an analogous result over bounded domains.

▶ **Corollary 12.** *Let $\mathbb{T} = [0, N)$ be a bounded interval of reals. Given an* MSO(<)*-formula $\varphi(\overline{P}, \overline{X}, \overline{Y})$ one can compute a sentence $\theta(\overline{P})$ and formula $\psi(\overline{P}, \overline{X}, \overline{Y})$ such that for every structure $\mathcal{M} = \langle \mathbb{T}, <, \overline{\mathbf{P}} \rangle$, φ has a causal uniformizer over $\mathcal{M}$ if and only if $\mathcal{M} \models \theta$ and in this case ψ is such a causal uniformizer.*

We now seek to apply this result to formulas of MSO$(<, +1)$ by first removing all references to the $+1$ relation using the following translation.

5.1 Eliminating the Metric

Given an MSO$(<, +1)$-formula φ, we define a straightforward syntactic transformation into an MSO$(<)$-formula $\overline{\varphi}$ such that there is a natural bijection between models of φ with domain $[0, N)$ and models of $\overline{\varphi}$ with domain $[0, 1)$.

With each monadic predicate X that appears in φ, we associate a collection $X_0, \ldots, X_{N-1}$ of N fresh monadic predicates. Intuitively, each X_i is a predicate on $[0, 1)$ that represents X over the subinterval $[i, i + 1)$. Formally, an interpretation of X over domain $[0, N)$ yields interpretations of the X_i over $[0, 1)$ by defining $X_i(t)$ if and only if $X(i + t)$. Note that this correspondence yields a bijection between interpretations of X on $[0, N)$ and interpretations of $X_0, \ldots, X_{N-1}$ on $[0, 1)$.

We can assume that φ does not contain any (first- or second-order) existential quantifiers, by replacing them with combinations of universal quantifiers and negations if need be. It is also convenient to rewrite φ into a formula that makes use of a unary function $+1$ instead of the $+1$ relation. To this end, replace every occurrence of $+1(x, y)$ in φ by $(x < N - 1 \wedge x + 1 = y)$.

Next, replace every instance of $\forall x\, \psi$ in φ by the formula

$$\forall x \left(\psi[x/x] \wedge \psi[x + 1/x] \wedge \ldots \wedge \psi[x + (N - 1)/x]\right),$$

where $\psi[t/x]$ denotes the formula resulting from substituting every free occurrence of the variable x in ψ by the term t. Intuitively, this transformation is legitimate since first-order variables in our target formula will range over $[0, 1)$ rather than $[0, N)$.

Having carried out these substitutions, use simple arithmetic to rewrite every term in φ as $x + k$, where x is a variable and $k \in \mathbb{N}$ is a non-negative integer constant.

Every inequality occurring in φ is now of the form $x + k < N - 1$ or $x + k_1 < y + k_2$. Replace every inequality of the first kind by **true** if $k + 2 \leq N$ and by **false** otherwise, and replace every inequality of the second kind by (i) $x < y$, if $k_1 = k_2$; (ii) **true**, if $k_1 < k_2$; and (iii) **false** otherwise.

Every equality occurring in φ is now of the form $x + k_1 = y + k_2$. Replace every such equality by $x = y$ if $k_1 = k_2$, and by **false** otherwise.

Every use of monadic predicates in φ now has the form $X(x + k)$, for $k \leq N - 1$. Replace every such predicate by $X_k(x)$.

Finally, replace every occurrence of $\forall X\, \psi$ in φ by $\forall X_0\, \forall X_1\, \ldots \forall X_{N-1}\, \psi$. The resulting formula is the desired $\overline{\varphi}$. Note that $\overline{\varphi}$ does not mention the $+1$ function, and is therefore indeed a non-metric (i.e., purely order-theoretic) sentence in MSO$(<)$. The following proposition is then clear.

▶ **Proposition 13** ([9])**.** *$\langle [0, N), <, +1, \mathbf{P} \rangle \models \varphi$ if and only if $\langle [0, 1), <, \mathbf{P}_0, \ldots, \mathbf{P}_{N-1} \rangle \models \overline{\varphi}$.*

5.2 Main Result

The following result, concerning the computability of uniformizers in MSO$(<, +1)$, is the main result of the paper. We state this problem for unlabelled intervals, although considering

labelled intervals is essential in the proof. Our proof technique generalises straightforwardly to handle a more general result involving uniformisation of $\mathsf{MSO}(<,+1)$ over labelled intervals [6].

▶ **Theorem 3.** *Let $\mathbb{T} = [0, N)$ be a bounded interval of reals. Given an $\mathsf{MSO}(<,+1)$-formula $\varphi(\overline{X}, \overline{Y})$ one can decide whether φ has an $\mathsf{MSO}(<,+1)$-definable causal uniformizer over $\langle \mathbb{T}, <, +1\rangle$ and if so one can compute such a uniformizer $\psi(\overline{X}, \overline{Y})$.*

Proof. To simplify notation, we consider the special case where φ has only X and Y as free variables.

Step 1. Applying the transformation described in Section 5.1 to $\varphi(X, Y)$ yields an $\mathsf{MSO}(<)$-formula $\overline{\varphi}(X_0, \ldots, X_{N-1}, Y_0, \ldots, Y_{N-1})$, such that there is a natural bijection between the models of φ over $[0, N)$ and the models of $\overline{\varphi}$ over $[0, 1)$, where $X_i(t)$ holds if and only if $X(i + t)$ holds for $i = 0, 1, \ldots, N - 1$ and $0 \leq t < 1$.

Step 2. We reduce the problem of uniformizing φ over $\langle \mathbb{T}, <, +1\rangle$ to an N-*phase* uniformization procedure applied to $\overline{\varphi}$. In the first phase, we construct a causal operator to determine the values of Y_0 from the values of X_0. The second phase then constructs a causal operator to determine the values of Y_1 from those of X_1, treating the values of X_0 and Y_0 generated in the previous phase as parameters that are already fixed. At the end of the N-th phase we wish $\overline{\varphi}$ to be satisfied by the values of $\overline{X}$ and $\overline{Y}$ we have determined. Our claim is that we can construct a series of functions in such a scenario if and only if we can uniformize φ over $\langle \mathbb{T}, <, +1\rangle$.

We formalise the above idea by defining N uniformization problems $G_0, G_1 \ldots, G_{N-1}$ involving only $\mathsf{MSO}(<)$ over $[0, 1)$. The basic data of each problem G_k, $0 \leq k < N$ are illustrated in Figure 1.

We define the problems G_k by backward induction, starting with G_{N-1}. The winning condition in this problem is

$$\varphi_{N-1}(X_0, \ldots, X_{N-1}, Y_0, \ldots, Y_{N-1}) \stackrel{\text{def}}{=} \overline{\varphi}(X_0, \ldots, X_{N-1}, Y_0, \ldots, Y_{N-1}),$$

where $X_0, \ldots, X_{N-2}$ and $Y_0, \ldots, Y_{N-2}$ are considered as parameters and X_{N-1} and Y_{N-1} as variables. Applying Corollary 12 we obtain a domain formula θ_{N-1} and uniformizer ψ_{N-1} for φ_{N-1}.

Suppose that we have defined G_k, with basic data as given in Figure 1. Then we define G_{k-1} as follows. The formula to be uniformized, denoted φ_{k-1}, is defined to be the domain formula θ_k from the preceding problem G_k. In G_{k-1} we consider $X_0, \ldots, X_{k-2}$ and $Y_0, \ldots, Y_{k-2}$ as parameters and X_{k-1} and Y_{k-1} as variables. The domain formula θ_{k-1} and uniformizer ψ_{k-1} are then obtained by applying Corollary 12 to the problem G_{k-1}.

Notice that the domain formula θ_0 is equivalent to either the sentence **true** or the sentence **false**. Below we show that $\theta_0 \equiv$ **true** just in case $\varphi(X, Y)$ has a uniformizer over $\langle \mathbb{T}, <, +1\rangle$.

Step 3: *Definition of uniformizer $\psi(X, Y)$ for $\varphi(X, Y)$.* Let

$$\overline{\psi} \stackrel{\text{def}}{=} \psi_0 \wedge \cdots \wedge \psi_{N-1}.$$

Then $\overline{\psi}$ is a formula in variables $X_0, \ldots, X_{N-1}$ and $Y_0, \ldots, Y_{N-1}$. The $\mathsf{MSO}(<,+1)$-formula $\psi(X, Y)$ is obtained from $\overline{\psi}$ by replacing every occurrence of $X_i(t)$ with $X(i + t)$ and $Y_i(t)$ with $Y(i+t)$. The transformation from $\overline{\psi}$ to ψ can be seen as the reverse of the transformation in Section 5.1, motivating our choice of notation.

This completes the description of the procedure to decide whether $\varphi(X, Y)$ has a uniformizer and if so to construct such a uniformizer. We turn now to the correctness of this construction.

Uniformization Problem G_k	
Input:	
Winning condition	$\varphi_k(X_0, \ldots, X_k, Y_0, \ldots, Y_k)$
Parameters	$X_0, \ldots, X_{k-1}, Y_0, \ldots, Y_{k-1}$
Variables	X_k, Y_k
Output:	
Domain formula	$\theta_k(X_0, \ldots, X_{k-1}, Y_0, \ldots, Y_{k-1})$
Uniformizer	$\psi_k(X_0, \ldots, X_k, Y_0, \ldots, Y_k)$

Figure 1 Basic data for G_k.

Suppose that $\theta_0 \equiv \mathbf{true}$. We show that $\psi(X, Y)$ defines a causal uniformizer for $\varphi(\overline{X}, \overline{Y})$ on $\langle \mathbb{T}, <, +1 \rangle$.

Let $\mathbf{X} \subseteq \mathbb{T}$. We must show that there is a unique $\mathbf{Y} \subseteq \mathbb{T}$ such that $\psi(\mathbf{X}, \mathbf{Y})$ and for this $\mathbf{Y}$ also $\varphi(\mathbf{X}, \mathbf{Y})$. Write $\overline{\mathbf{X}} = \mathbf{X}_0, \ldots, \mathbf{X}_{N-1}$ for the tuple of subsets of $[0, 1)$ defined by $\mathbf{X}_i(t)$ if and only if $\mathbf{X}(i + t)$.

Since $\theta_0 \equiv \mathbf{true}$ we know that ψ_0 uniformizes φ_0. Thus there exists $\mathbf{Y}_0 \subseteq [0, 1)$ such that $\psi_0(\mathbf{X}_0, \mathbf{Y}_0)$ and $\varphi_0(\mathbf{X}_0, \mathbf{Y}_0)$ both hold. But $\theta_1 \equiv \varphi_0$, so $\theta_1(\mathbf{X}_0, \mathbf{Y}_0)$ also holds. Since ψ_1 uniformizes φ_1 there exists $\mathbf{Y}_1 \subseteq [0, 1)$ such that $\psi_1(\mathbf{X}_0, \mathbf{X}_1, \mathbf{Y}_0, \mathbf{Y}_1)$ and $\varphi_1(\mathbf{X}_0, \mathbf{X}_1, \mathbf{Y}_0, \mathbf{Y}_1)$ both hold. Continuing in this vein we successively generate predicates $\mathbf{Y}_0, \ldots, \mathbf{Y}_{N-1}$ such that

$$\psi_0(\mathbf{X}_0, \mathbf{Y}_0) \wedge \psi_1(\mathbf{X}_0, \mathbf{X}_1, \mathbf{Y}_0, \mathbf{Y}_1) \wedge \cdots \wedge \psi_{N-1}(\mathbf{X}_0, \ldots, \mathbf{X}_{N-1}, \mathbf{Y}_0, \ldots, \mathbf{Y}_{N-1}) .$$

Thus by definition of $\overline{\psi}$ we have $\overline{\psi}(\mathbf{X}_0, \ldots, \mathbf{X}_{N-1}, \mathbf{Y}_0, \ldots, \mathbf{Y}_{N-1})$.

Now define $\mathbf{Y} \subseteq \mathbb{T}$ by having $\mathbf{Y}(t + i)$ hold if and only if $\mathbf{Y}_i(t)$ holds for $i = 0, \ldots, N - 1$ and $0 \leq t < 1$. Then by definition of ψ we have $\psi(\mathbf{X}, \mathbf{Y})$. Furthermore, since ψ_{N-1} uniformizes φ_{N-1} we also have that

$$\varphi_{N-1}(\mathbf{X}_0, \ldots, \mathbf{X}_{N-1}, \mathbf{Y}_0, \ldots, \mathbf{Y}_{N-1}) .$$

But φ_{N-1} was defined to be $\overline{\varphi}$ and thus by Proposition 13 we have that $\varphi(\mathbf{X}, \mathbf{Y})$ also holds.

The fact that ψ is functional and causal can easily be obtained from the corresponding properties of $\psi_0, \ldots, \psi_{N-1}$ in the above construction. Reversing the above argument also allows us to deduce that if $\varphi(X, Y)$ has a uniformizer over $\langle \mathbb{T}, <, +1 \rangle$ then $\theta_0 \equiv \mathbf{true}$: given a uniformizer $\psi(X, Y)$ for φ one successively generates uniformizers for φ_{N-1} down to φ_0. ◀

6 Lower Bounds

Define a family of functions $\exp_k : \mathbb{N} \to \mathbb{N}$ by $\exp_0(n) = n$ and $\exp_{k+1}(n) = 2^{\exp_k(n)}$. A function $f : \mathbb{N} \to \mathbb{N}$ is *non-elementary* if it grows faster than any $\exp_k$.

The procedure for uniformizing $\mathsf{MSO}(<, +1)$-formulas over bounded time domain $\mathbb{T} = [0, N)$ described in Section 5.2 has non-elementary complexity. This blow-up arises not only from the non-elementary transformation of $\mathsf{MSO}(<, +1)$ to automata—repeated application of Corollary 12 leads to an N-fold exponential blow-up.

In this section we give a non-elementary lower bound for the bounded uniformization problem for $\mathsf{FO}(<, +1)$ that holds even for formulas of a fixed quantifier alternation depth (for

which satisfiability over bounded intervals is elementary[1]). This is proven by reduction from the language emptiness problem for star-free regular expressions. The construction we outline below can also be used to show that uniformization is non-elementary also for the temporal logic MTL for which satisfiability over bounded intervals is EXPSPACE-complete [9]. We have used a related idea to show that the language emptiness problem for alternating timed automata over bounded time domains is non-elementary [7].

A *star-free regular expression* over alphabet Σ is built from the symbols $\emptyset$ and σ, for any $\sigma \in \Sigma$, using the operations of union $(+)$, concatenation $(\cdot)$, and complementation $(\neg)$. Such an expression E denotes a language $L(E) \subseteq \Sigma^*$ which is defined as follows:

$L(\emptyset) = \emptyset$ and $L(\sigma) = \{\sigma\}$;
$L(\neg E) = \Sigma^* \setminus L(E)$;
$L(E + E') = L(E) \cup L(E')$;
$L(E \cdot E') = L(E) \cdot L(E')$.

The *operator depth* $\mathrm{odp}(E)$ of a star-free regular expression E is defined as follows:

$\mathrm{odp}(\emptyset) = \mathrm{odp}(\sigma) = 1$;
$\mathrm{odp}(\neg E) = \mathrm{odp}(E)$;
$\mathrm{odp}(E + E') = \max\{\mathrm{odp}(E), \mathrm{odp}(E')\} + 1$;
$\mathrm{odp}(E \cdot E') = \max\{\mathrm{odp}(E), \mathrm{odp}(E')\} + 1$.

Note that negation does not count toward the operator depth.

The following result was shown in [14].

▶ **Theorem 14.** *The language emptiness problem for star-free regular expressions is non-elementary.*

Given a star-free regular expression E over alphabet Σ and a word $w = w_0 w_1 \dots w_{n-1} \in \Sigma^*$ we define the *membership game* $\mathbb{G}(w, E)$. This is a two-player game with N rounds, where N is the operator depth of E. The two players are *Prover*, who is trying to show $w \in E$, and *Refuter*, who is trying to show $w \notin E$. The positions of the game are triples (b, e, F) where b and e are positions in the word w and F has the form G or $\neg G$ for G a sub-expression of E. The initial position is $(0, n, E)$. If the position at the start of a given round is (b, e, F) the goal of Prover is to show that $w_b w_{b+1} \dots w_{e-1} \in F$. The round proceeds as follows:

- If $F \equiv F_1 \cdot F_2$ then Prover moves first by choosing an index i with $b \le i \le e$. Refuter responds by selecting either (b, i, F_1) or (i, e, F_2) as the position in the next round;
- If $F \equiv \neg(F_1 \cdot F_2)$ then Refuter moves first by choosing an index i with $b \le i \le e$. Prover responds by selecting either $(b, i, \neg F_1)$ or $(i, e, \neg F_2)$ as the position in the next round;
- If $F \equiv F_1 + F_2$ then Prover selects either (b, e, F_1) or (b, e, F_2) as the position in the next round;
- If $F \equiv \neg(F_1 + F_2)$ then Refuter selects either $(b, e, \neg F_1)$ or $(b, e, \neg F_2)$ as the position in the next round.

The positions (b, e, σ), $(b, e, \neg\sigma)$, $(b, e, \emptyset)$ and $(b, e, \neg\emptyset)$ are terminal and are classified as winning for Prover or Refuter according to whether $u_b, u_{b+1} \dots, u_{e-1}$ is a member of the corresponding expression.

It is clear that Prover has a winning strategy in $\mathbb{G}(w, E)$ if and only if $w \in L(E)$.

For any regular expression F, let $Sub(F)$ be the set of sub-expressions of E along with their negations. Given that a position in $\mathbb{G}(w, E)$ is a triple from the set $\Pi \stackrel{\text{def}}{=} \{0, \dots, n\}^2 \times Sub(E)$,

[1] The paper [9] provides an elementary reduction of the satisfiability problem for $\mathsf{FO}(<, +1)$ over bounded intervals to the problem for $\mathsf{FO}(<)$ which does not change the quantifier alternation depth.

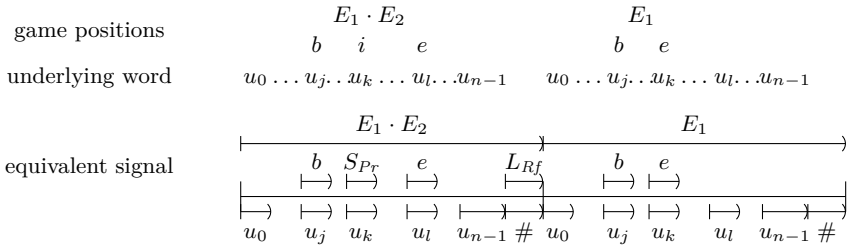


Figure 2 A signal encoding a play

a *play* of $\mathbb{G}(w, E)$ can be represented as a word in Π^* denoting a sequence of successive positions. The idea of our reduction is to encode plays as signals over a domain $[0, N+1)$ and to construct a formula of $\mathsf{FO}(<, +1)$ that is satisfied by a signal if and only it encodes a winning play for Prover. In this encoding successive game positions are encoded in successive unit-length subintervals of the domain.

Our encoding represents plays using monadic predicates P_b, P_e, $P_\#$, $P_{S_{Pr}}$, $P_{L_{Pr}}$, $P_{R_{Pr}}$, $P_{S_{Rf}}$, $P_{L_{Rf}}$, $P_{R_{Rf}}$ and two families of predicates P_σ, $\sigma \in \Sigma$ and P_F, $F \in \mathit{Sub}(E)$. For a signal to encode a play of $\mathbb{G}(w, E)$ we require, among other things, that:

- The predicates P_F, $F \in \mathit{Sub}(E)$, hold on intervals $[k, k+1)$ for $k = 0, 1, \ldots, N$ and are mutually exclusive.
- Exactly one of the predicates P_σ, $\sigma \in \Sigma$, and $P_\#$ holds at any given point. Moreover these predicates hold in sequence $P_{w_0}, P_{w_1}, \ldots, P_{w_{n-1}}, P_\#$ over the interval $[0, 1)$.
- If $s = t + 1$ then P_σ holds at s if and only if P_σ holds at t; likewise $P_\#$ holds at s if and only if $P_\#$ holds at t.
- In each successive unit interval $[k, k+1)$ the predicate P_b holds in one sub-interval over which some predicate P_σ or $P_\#$ also holds. The same restriction applies to P_e.
- If $P_{E_1 \cdot E_2}$ holds on $[k, k+1)$ then $P_{S_{Pr}}$ holds in exactly one sub-interval in this time unit and either $P_{L_{Rf}}$ holds at the same time as $P_\#$ during this time unit and P_{E_1} holds in the next time unit, or $P_{R_{Rf}}$ holds at the same time as $P_\#$ during this time unit and P_{E_2} holds in the next time unit.

Notice how the third clause ensures that a copy of the word w is propagated between successive time units, cf. Figure 2.

A game position (i, j, F) is encoded in a unit-length subinterval $[k, k+1)$ by having P_F hold throughout the interval, P_b hold at the same time as P_{w_i} and P_e hold at the same time as P_{w_j} (where we take $w_n = \#$). The idea is that the propositions $P_{S_{Pr}}$, $P_{L_{Pr}}$ and $P_{R_{Pr}}$ encode moves of Prover and the propositions $P_{S_{Rf}}$, $P_{L_{Rf}}$ and $P_{R_{Rf}}$ encodes moves of Refuter; the respective position of these S propositions indicate the position around which the input word is split while the L and R propositions indicate whether that player wished to continue playing in the left or right subword.

Given a star-free regular expression E we can define a formula $\varphi_E(\overline{X}, \overline{Y})$ such that φ_E has a uniformizer if and only if there exists a word $w \in \Sigma^*$ such that Prover has a winning strategy in the game $\mathbb{G}(w, E)$. The tuple $\overline{X}$ just includes the predicates $P_{S_{Rf}}$, $P_{L_{Rf}}$ and $P_{R_{Rf}}$ while the tuple $\overline{Y}$ includes all the other predicates mentioned above. We define φ_E such that it is true on any signal that represents a play of $\mathbb{G}(w, E)$ that is winning for Prover according to the encoding defined above. For signals that do not encode such plays φ_E is only satisfied if the predicates $P_{S_{Rf}}$, $P_{L_{Rf}}$ or $P_{R_{Rf}}$ do not obey the above rules (intuitively Refuter broke the rules of the game). Details of this encoding can be found in [6].

▶ **Theorem 15.** *The time-bounded uniformization problem for* $\mathsf{FO}(<,+1)$ *is non-elementary for formulas of quantifier alternation depth at most three.*

7 Conclusion

In this paper, we considered extensions of Church's synthesis problem to the continuous-time domain of the reals. We proved that under the finite-variability and right-continuous assumption, Church's problem is decidable when we require that the uniformizer be definable in the same logic as the specification. This result holds over unbounded intervals when only the $<$ relation is available and over every fixed bounded-length interval when the $+1$ relation is also used.

References

1. J. R. Büchi. On a decision method in restricted second-order arithmetic. In *Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.
2. J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the AMS*, 138(27):295–311, 1969.
3. A. Church. Applications of recursive arithmetic to the problem of circuit synthesis. In *Summaries of the Summer Institute of Symbolic Logic*, volume 1, pages 3–50. Cornel Univ, Ithaca, 1957.
4. P. Hänsch, M. Slaats, and W. Thomas. Parametrized regular infinite games and higher-order pushdown strategies. In *Proceedings of FCT 09*, volume 5699 of *Lecture Notes in Computer Science*. Springer, 2009.
5. Y. Hirshfeld and A. Rabinovich. Logics for real time: Decidability and complexity. *Fundam. Inform.*, 62(1), 2004.
6. M. Jenkins, J. Ouaknine, A. Rabinovich, and J. Worrell. The Church synthesis problem with metric (full version). `http://www.cs.ox.ac.uk/people/mark.jenkins/church-metric.pdf`.
7. M. Jenkins, J. Ouaknine, A. Rabinovich, and J. Worrell. Alternating timed automata over bounded time. In *LICS*, pages 60–69. IEEE Computer Society, 2010.
8. S. Lifsches and S. Shelah. Uniformization and Skolem functions in the class of trees. *J. Symb. Log.*, 63(1):103–127, 1998.
9. J. Ouaknine, A. Rabinovich, and J. Worrell. Time-bounded verification. In *Proceedings of CONCUR 09*, volume 5710 of *Lecture Notes in Computer Science*. Springer, 2009.
10. A. Rabinovich. Finite variability interpretation of monadic logic of order. *Theoretical Computer Science*, 275(1-2):111–125, 2002.
11. A. Rabinovich. The Church synthesis problem with parameters. *Logical Methods in Computer Science*, 3(4), 2007.
12. A. Rabinovich. On decidability of monadic logic of order over the naturals extended by monadic predicates. *Inf. Comput.*, 205(6):870–889, 2007.
13. S. Shelah. The monadic theory of order. *The Annals of Mathematics*, 102(3):379–419, 1975.
14. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proceedings of STOC 73*, pages 1–9, New York, NY, USA, 1973. ACM.

A Pumping Lemma for Collapsible Pushdown Graphs of Level 2*

Alexander Kartzow

Universität Leipzig, Institut für Informatik
Johannisgasse 26, 04103 Leipzig

Abstract

We present a pumping lemma for the class of collapsible pushdown graphs of level 2. This pumping lemma even applies to the ε-contractions of level 2 collapsible pushdown graphs. Our pumping lemma also improves the bounds of Hayashi's pumping lemma for indexed languages.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Collapsible pushdown graph, ε-contraction, pumping lemma

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.322

1 Introduction

Recently, generalisations of pushdown systems have gained attention for the verification of higher-order functional programs. This stems from the fact that collapsible higher-order pushdown systems generate exactly the same trees as higher-order recursion schemes [6]. Recursion Schemes can be fruitfully applied in the verification of functional programs [11]. The correspondence between collapsible pushdown trees and higher-order recursion schemes improves a result of Knapik et al. [10] showing that higher-order pushdown systems generate the same trees as *safe* higher-order recursion schemes. Safety is a syntactic condition whose semantical status is still open: it is conjectured that there is a recursion scheme that generates a tree which is not generated by any safe scheme.[1]

Also from a model-theoretic point of view, the classes of collapsible and higher-order pushdown systems are interesting. Carayol and Wöhrle [3] proved that the ε-contractions of graphs generated by higher-order pushdown systems are exactly the graphs in the Caucal-hierarchy. Thus, all these graphs have decidable monadic second-order theories. Collapsible pushdown graphs display a rather different behaviour: even on the second level of the hierarchy, there is a graph with undecidable monadic second-order theory. Nevertheless, the first-order model checking on level 2 collapsible pushdown graphs is decidable because all these graphs are tree-automatic [8]. Recently, Broadbent [2] proved that even first-order logic is undecidable on the collapsible pushdown graphs of level 3. Moreover, Hague et al. [6] showed that the modal μ-calculus is decidable on all collapsible pushdown graphs. These decidability results give collapsible pushdown graphs a unique status among natural classes of graphs. There is (almost[2]) no other known natural class of graphs with decidable modal μ-calculus model checking but undecidable monadic second-order theories.

* This research was carried out while the author was working at TU Darmstadt and funded by the DFG grant OT 147/5-1

[1] Recently, Parys [13] showed that the uniform safety conjecture is true: there is a level 2 recursion scheme which generates a tree that is not generated by any safe level 2 scheme

[2] Nested pushdown trees share the same status. But these form in some sense a subclass of collapsible pushdown graphs (cf. [9])

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 322–336

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

Even though (collapsible) pushdown systems generate important classes of graphs, useful characterisations of their structure are still rare. We [8] showed that the second-level of the collapsible pushdown graph hierarchy is tree-automatic. Nevertheless, we still miss techniques for disproving membership in the collapsible pushdown hierarchy. In classical automata-theory, pumping lemmas form a good tool for disproving membership in languages defined by finite automata or pushdown systems. On higher levels, similar results are still missing. The only results in this direction that are known to the author are a pumping lemma of Hayashi [7] and a shrinking lemma of Gilman [5], both for indexed languages. Since the class of string-languages accepted by the second level of the collapsible higher-order pushdown hierarchy is the class of indexed languages, this is a first step towards pumping on higher-order pushdown systems. Blumensath [1] published an extension of this pumping lemma to all levels of the higher-order pushdown hierarchy. Unfortunately, there is an irrecoverable error in his proof (cf. [12]). Thus, the question for pumping lemmas for higher-order pushdown systems of level at least 3 is still open. Moreover, even for the second level of collapsible pushdown graphs no pumping lemma was known so far.

In this paper, we close the latter gap. As already mentioned, collapsible pushdown graphs are tree-automatic, i.e., there is an encoding of configurations in trees such that a (finite tree-)automaton accepts the encodings of configurations reachable form the initial one. Of course, the regular pumping lemma applies to this finite automaton. Since accepting runs of this automaton encode runs of the collapsible pushdown system, the trees obtained by regular pumping can be turned into runs of the collapsible pushdown system. Thus, the existence of a large configuration in the collapsible pushdown graph implies the existence of infinitely many runs. A refinement of this argument yields a pumping lemma for all ε-contractions of collapsible pushdown graphs of level 2.

1.1 Outline of the paper

In the next section, we recall the standard notion of trees, finite tree-automata and the pumping lemma for finite tree-automata. At the end of that section we introduce the general approach how tree-automaticity of the reachability predicate can be turned into a pumping lemma. In Section 3 we present the notion of level 2 collapsible pushdown graphs. Furthermore, we recall the main result from [8], i.e., we present an encoding of configurations of 2-CPG in trees that turns the set of reachable configurations into a regular set of trees. We also recall what an automaton determining the reachable configurations looks like. In Section 4 we compute the specific bounds of the pumping lemma for level 2 collapsible pushdown graphs obtained from our general approach. We then refine this result in Section 5 to ε-contractions of such graphs. In Section 6 we apply our pumping lemma and prove that certain trees are not ε-contractions of any 2-CPG. Section 7 contains concluding remarks and points to open questions.

2 A Pumping Lemma for Structures with Automatic Reachability Relation

A Σ-*labelled tree* is a function $T : D \to \Sigma$ for a finite set $D \subseteq \{0,1\}^*$ which is closed under prefixes. For $d \in D$ we denote by T_d the *subtree rooted at* d. It is useful to define trees inductively by describing their left and right subtrees. For this purpose we fix the following notation. Let $\hat{T}$ and T' be Σ-labelled trees and $\sigma \in \Sigma$. Then we write $T := \sigma(\hat{T}, T')$ for the

Σ-labelled tree T with the following three properties

1. $T(\varepsilon) = \sigma$, 2. $T_0 = \hat{T}$, and 3. $T_1 = T'$.

Finally, let $\mathrm{dp}(T) := \max\{|d| : d \in \mathrm{dom}(T)\} + 1$ be the *depth* of T. A straightforward induction gives a bound on the number of nodes of a tree of a fixed depth.

▶ **Lemma 1.** *If T is a tree of depth d, then $|dom(T)| \leq 2^d - 1$.*

▶ **Corollary 2.** *There are at most $(|\Sigma| + 1)^{2^d - 1}$ many Σ-labelled trees of depth d.*

We briefly recall the notion of a (finite tree-) automaton and the pumping lemma for these automata.

▶ **Definition 3.** A *(finite tree-)automaton* is a tuple $\mathcal{A} = (\Sigma, Q, \Delta, q_I)$ where Σ is a finite alphabet, Q a finite set of states with a distinguished symbol $\perp \in Q$, $\Delta \subseteq Q \times \Sigma \times Q \times Q$ a transition relation and $q_I \in Q$ the initial state. An *accepting run* of $\mathcal{A}$ on a Σ-labelled tree T is a map $\rho : \{0,1\}^* \to Q$ such that

1. $(\rho(d), T(d), \rho(d0), \rho(d1)) \in \Delta$ for all $d \in \mathrm{dom}(T)$ and
2. $\rho(\varepsilon) = q_I$ and $\rho(d) = \perp$ for all $d \in \{0,1\}^* \setminus \mathrm{dom}(T)$.

The *language* accepted by $\mathcal{A}$ is $L(\mathcal{A}) := \{T : \exists \text{ accepting run of } \mathcal{A} \text{ on } T\}$.

In order to develop a pumping lemma for structures with automatic reachability relation, we will use the pumping lemma for regular languages.

▶ **Lemma 4** (see [4])**.** *Let $\mathcal{A} = (\Sigma, Q, \Delta, q_I)$ be an automaton recognising the (tree-)language L, let $T \in L$ and let ρ be an accepting run of $\mathcal{A}$ on T. If $d \in dom(T)$ with $\mathrm{dp}(T_d) > |Q|$, then there are nodes $d \leq d_1 \leq d_2 \in dom(T)$ such that the following holds. If we replace in t the subtree rooted at d_1 by the subtree rooted at d_2, the tree T_0 resulting from this replacement satisfies $T_0 \in L$. Furthermore, let $T_1, T_2, T_3, \ldots$ be the infinite sequence of trees where $T_1 = T$ and T_{i+1} arises from T_i by replacing the subtree rooted at d_2 in T_i by the subtree rooted at d_1 in T_i, then $T_i \in L$ for all $i \in \mathbb{N}$. Furthermore, for each $i \in \mathbb{N}$ there is an accepting run ρ_i on T_i that coincides with ρ on all positions except for those in the subtree induced by d.*

In order to define the notion of an automatic reachability relation, we recall the definition of the *convolution* of two Σ-labelled trees T and T'. This is the tree

$$T \otimes T' : \mathrm{dom}(T) \cup \mathrm{dom}(T') \to (\Sigma \cup \{\Box\})^2$$

$$(T \otimes T')(d) := \begin{cases} (T(d), T'(d)) & \text{if } d \in \mathrm{dom}(T) \cap \mathrm{dom}(T') \\ (T(d), \Box) & \text{if } d \in \mathrm{dom}(T) \setminus \mathrm{dom}(T') \\ (\Box, T'(d)) & \text{if } d \in \mathrm{dom}(T') \setminus \mathrm{dom}(T) \end{cases}$$

where $\Box$ is a new symbol for padding. The convolution of trees is the standard concept if one wants to use a finite tree automaton for recognising some relation on trees. We say $\mathcal{A}$ recognises some relation R if $L(\mathcal{A}) := \{T_1 \otimes T_2 \otimes \cdots \otimes T_n : (T_1, T_2, \ldots, T_n) \in R\}$. In this case we say that R is automatic.

Using the regular pumping lemma, we obtain the following two pumping lemmas for structures with automatic reachability relation.

▶ **Lemma 5.** *Let $\mathfrak{G} = (D, \vdash)$ be some graph such that D is a regular set of trees over the alphabet Σ. Assume that the reachability relation R on $\mathfrak{G}$ is recognised by some tree-automaton with q states. If there starts a finite path p at $d \in D$ of length $(|\Sigma| + 1)^{2^{(\mathrm{dp}(d)+q)}}$ then there start infinitely many paths at d.*

Proof. If p visits some vertex $d' \in D$ twice then there is obviously an infinite path starting at d and passing d' infinitely many times.

Otherwise, p passes $(|\Sigma|+1)^{2^{(\mathrm{dp}(d)+q)}}$ many different trees. Due to Corollary 2, p passes some d' of depth $\mathrm{dp}(d') > \mathrm{dp}(d)+q$. Since $\mathrm{dp}(d') - \mathrm{dp}(d) > q$, we may apply the regular pumping lemma to $d \otimes d'$ in such a way that we obtain infinitely many trees $d_1, d_2, \ldots$ such that $(d, d_i) \in R$. This means that for each i there is a path from d to d_i. ◀

▶ **Lemma 6.** *Let $\mathfrak{G} = (D, \vdash)$ be some graph such that D is a regular set of trees over the alphabet Σ. Assume that the reachability relation R on $\mathfrak{G}$ is recognised by some tree-automaton with q states. If $|\{x \in D : (d,x) \in R\}| > (|\Sigma|+1)^{2^{(\mathrm{dp}(d)+q)}}$, then $\{x \in D : (d,x) \in R\}$ is an infinite set.*

Proof. Assume that $|\{x \in D : (d,x) \in R\}| > (|\Sigma|+1)^{2^{(\mathrm{dp}(d)+q)}}$ There must be some $d' \in D$ such that $(d,d') \in R$ and $\mathrm{dp}(d') > \mathrm{dp}(d)+q$. We conclude as in the previous lemma. ◀

In the rest of this paper, we develop an adaptation of these lemmas to collapsible pushdown graphs of level 2.

3 Collapsible Pushdown Systems and Graphs

In this section we define our notation of collapsible pushdown systems (of level 2). For a more detailed introduction, we refer the reader to [6] or [9]. Afterwards, we present those results from [8] that are relevant for the results of this paper.

3.1 Collapsible Pushdown Stacks of Level 2

First, we provide some terminology concerning stacks of (collapsible) higher-order pushdown systems. We write Σ^{*2} for $(\Sigma^*)^*$ and Σ^{+2} for $(\Sigma^+)^+$. We call an $s \in \Sigma^{*2}$ a 2-*word*.

Let us fix a 2-word $s \in \Sigma^{*2}$ which consists of an ordered list $w_1, w_2, \ldots, w_m \in \Sigma^*$. We separate the words of this list by colons writing $s = w_1 : w_2 : \ldots : w_m$. We write $\mathrm{wdt}(s) := m$ for the *width* of s and $\mathrm{hgt}(s) := \max\{|w_i| : 1 \leq i \leq m\}$ for the *height* of s. For a second word $s' = w'_1 : w'_2 : \ldots : w'_n \in \Sigma^{*2}$, we write $s : s'$ for the concatenation $w_1 : w_2 : \ldots : w_m : w'_1 : w'_2 : \ldots : w'_n$. If $w \in \Sigma^*$, we write $[w]$ for the 2-word that consists of a list of one word which is w.

A level 2 collapsible pushdown stack is a special element of $(\Sigma \times \{1,2\} \times \mathbb{N})^{+2}$ that is generated by certain stack operations from an initial stack. We introduce these in the following definitions. The natural numbers following the stack symbol represent a pointer: every element in a collapsible pushdown stack has a pointer to some substack and applying the collapse operation returns the substack to which the topmost symbol of the stack points. Here, the first number denotes the *collapse level*. If it is 1 the collapse pointer always points to the symbol below the topmost symbol and the collapse operations just removes the topmost symbol. The more interesting case is when the collapse level of the topmost symbol of the stack s is 2. Then the stack obtained by the collapse contains the first n words of s where n is the second number in the topmost element of s.

The initial level 1 stack is $\bot_1 := (\bot, 1, 0)$ and the initial level 2 stack is $\bot_2 := [\bot_1]$.

Let $k \in \{1,2\}$ and let $s = w_1 : w_2 : \ldots : w_n \in (\Sigma \times \{1,2\} \times \mathbb{N})^{+2}$ be a 2-word such that $w_n = a_1 a_2 \ldots a_m$ with $a_i \in \Sigma \times \{1,2\} \times \mathbb{N}$ for all $1 \leq i \leq m$.

We define the *topmost* $(k-1)$*-word of* s as $\mathrm{top}_k(s) := \begin{cases} w_n & \text{if } k = 2 \\ a_m & \text{if } k = 1. \end{cases}$

- For $\mathrm{top}_1(s) = (\sigma, i, j) \in \Sigma \times \{1,2\} \times \mathbb{N}$, we define the *topmost symbol* $\mathrm{Sym}(s) := \sigma$, the *collapse-level of the topmost element* $\mathrm{CLvl}(s) := i$, and the *collapse-link of the topmost element* $\mathrm{CLnk}(s) := j$.

For s, w_n and k as before, $\sigma \in \Sigma \setminus \{\bot\}$, and $w'_n := a_1 \dots a_{m-1}$, we define the stack operations

$$\mathrm{pop}_k(s) := \begin{cases} w_1 : w_2 : \dots : w_{n-1} & \text{if } k = 2, n \geq 2, \\ w_1 : w_2 : \dots : w_{n-1} : w'_n & \text{if } k = 1, m \geq 2, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$$\mathrm{clone}_2(s) := w_1 : w_2 : \dots : w_{n-1} : w_n : w_n,$$

$$\mathrm{push}_{\sigma,k}(s) := \begin{cases} w_1 : w_2 : \dots : w_n(\sigma, 2, n-1) & \text{if k=2,} \\ w_1 : w_2 : \dots : w_n(\sigma, 1, m) & \text{if k=1,} \end{cases}$$

$$\mathrm{collapse}(s) := \begin{cases} w_1 : w_2 : \dots : w_{\mathrm{CLnk}(s)} & \text{if } \mathrm{CLvl}(s) = 2, n \geq \mathrm{CLnk}(s) > 0, \\ \mathrm{pop}_1(s) & \text{if } \mathrm{CLvl}(s) = 1, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The *set of level* 2*-operations* is $\mathrm{OP} := \{\mathrm{push}_{\sigma,1}, \mathrm{push}_{\sigma,2}, \mathrm{clone}_2, \mathrm{pop}_1, \mathrm{pop}_2, \mathrm{collapse}\}$. The *set of level* 2 *stacks*, $\mathrm{Stck}(\Sigma)$, is the smallest set that contains $\bot_2$ and is closed under all operations from OP.

Note that collapse- and pop_k-operations are only allowed if the resulting stack is a nonempty list of nonempty words. This avoids the special treatment of empty words or stacks. Furthermore, a collapse on level 2 summarises a non-empty sequence of pop_2-operations. For example, starting from $\bot_2$, we can apply a clone_2, a $\mathrm{push}_{\sigma,2}$, a clone_2, and finally a collapse. This sequence first creates a level 2 stack that contains 3 words and then performs the collapse and ends in the initial stack again. This example shows that clone_2-operations are responsible for the fact that collapse-operations on level 2 may remove more than one word from the stack. Since there is no level 1 clone operation, a collapse of level 1 always simulates exactly one pop_1.

For $s, s' \in \mathrm{Stck}(\Sigma)$, we call s' a *substack* of s if there are $n_1, n_2 \in \mathbb{N}$ such that $s' = {\mathrm{pop}_1}^{n_1}({\mathrm{pop}_2}^{n_2}(s))$. We write $s' \leq s$ if s' is a substack of s.

3.2 Collapsible Pushdown Systems and Collapsible Pushdown Graphs of Level 2

We introduce collapsible pushdown systems (CPS) and graphs (CPG) which are analogues of pushdown systems and pushdown graphs using collapsible pushdown stacks instead of ordinary stacks.

▶ **Definition 7.** A *collapsible pushdown system* is a tuple $\mathcal{S} = (\Sigma, Q, \Delta, q_0)$ where Σ is a finite stack alphabet with a special symbol $\bot \in \Sigma$, Q a finite set of states, $q_0 \in Q$ the initial state, and $\Delta \subseteq Q \times \Sigma \times Q \times \mathrm{OP}$ the transition relation.

For $q \in Q$ and $s \in \mathrm{Stck}(\Sigma)$ the pair (q, s) is called a *configuration*. We define labelled *transitions* on pairs of configurations by setting $(q_1, s) \vdash^\delta (q_2, t)$ if there is a $\delta = (q_1, \sigma, q_2, op) \in \Delta$ such that $\mathrm{Sym}(s) = \sigma$ and $op(s) = t$. The union of these relations is denoted by $\vdash := \bigcup_{\delta \in \Delta} \vdash^\delta$. We set $C(S)$ to be the set of all configurations that are reachable from $(q_0, \bot_2)$ via $\vdash$-paths. We call $C(S)$ the set of *reachable* configurations. The *collapsible pushdown graph generated by* S is $\mathrm{CPG}(S) := \big(C(S), C(S)^2 \cap \vdash\big)$

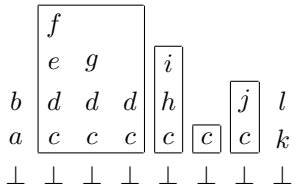


Figure 1 Example of blocks in a stack. These form a c-blockline.

▶ **Definition 8.** Let S be a CPS. A *run* ρ of S of length n is a function

$$\rho : \{0, 1, 2, \ldots, n\} \to Q \times (\Sigma \times \{1, 2\} \times \mathbb{N})^{*2} \text{ such that } \rho(0) \vdash \rho(1) \vdash \cdots \vdash \rho(n).$$

We write $\mathrm{len}(\rho) := n$ and call ρ a run from $\rho(0)$ to $\rho(n)$. We say ρ visits a stack s at i if $\rho(i) = (q, s)$ for some $q \in Q$.

For runs ρ, π of length n and m, respectively, with $\rho(n) = \pi(0)$, we define the composition $\rho \circ \pi$ of ρ and π in the obvious manner.

▶ Remark. Note that we do not require runs to start in the initial configuration.

From now on, we consider a fixed set of states Q and a fixed stack alphabet Σ with bottom-of-stack symbol $\bot$.

3.3 Encoding of Configurations as Trees

In [8] we proved that collapsible pushdown graphs are tree-automatic via an encoding function CEnc. We recall this function in this section. The concept underlying the encoding is that of blocks and blocklines. A blockline is a list of words that start with the same letter and a block is a list of words that start with the same two letters. We encode a blockline as follows. The root is labelled by the first letter of all words; for each block of the blockline, we add one subtree encoding the corresponding block. We present the details after the formal introduction of blocks and blocklines. For $w \in \Sigma^*$ and $s = w_1 : w_2 : \ldots : w_n \in \Sigma^{*2}$, we write $s' := w \setminus s$ for $s' = [ww_1] : [ww_2] : \ldots : [ww_n]$.

▶ **Definition 9** (γ-block(line))**.** For Γ some set and $\gamma \in \Gamma$, we call $b \in \Gamma^{*2}$ a *γ-block* if $b = [\gamma]$ or $b = \gamma\tau \setminus s'$ for some $\tau \in \Gamma$ and $s' \in \Gamma^{*2}$. See Figure 1 for examples of blocks. If $b_1, b_2, \ldots, b_n$ are γ-blocks, then we call $b_1 : b_2 : \ldots : b_n$ a *γ-blockline*.

Note that every stack forms a $(\bot, 1, 0)$-blockline. Furthermore, every blockline l decomposes uniquely as $l = b_1 : b_2 : \ldots : b_n$ of maximal blocks b_i. Another crucial observation is that a γ-block $b \in \Gamma^{*2} \setminus \Gamma$ decomposes as $b = \gamma \setminus l$ for some blockline l and we say l is the induced blockline of b. For $b \in \Gamma$ the induced blockline of $[b]$ is just the empty 2-word.

Now we encode a (σ, n, m)-blockline l in a tree by labelling the root with (σ, n), by encoding the blockline induced by the first block of l in the left subtree, and by encoding the rest of the blockline in the right subtree. In order to avoid repetitions, we do not repeat the symbol (σ, n) in the right subtree, but replace it by the default letter ε.

▶ **Definition 10.** Let $s = w_1 : w_2 : \ldots : w_n \in (\Sigma \times \{1, 2\} \times \mathbb{N})^{+2}$ be a (σ, l, k)-blockline. Let w'_i be words such that $s = (\sigma, l, k) \setminus (w'_1 : w'_2 : \ldots : w'_n)$. Set $s' := w'_1 : w'_2 : \ldots : w'_n$. As an abbreviation we write ${}_h s_i := w_h : w_{h+1} : \ldots : w_i$. Furthermore, let $w_1 : w_2 : \ldots : w_j$ be a maximal block of s. Note that $j > 1$ implies $w_{j'} = (\sigma, l, k)(\sigma', l', k')w''_{j'}$ for all $j' \leq j$, some

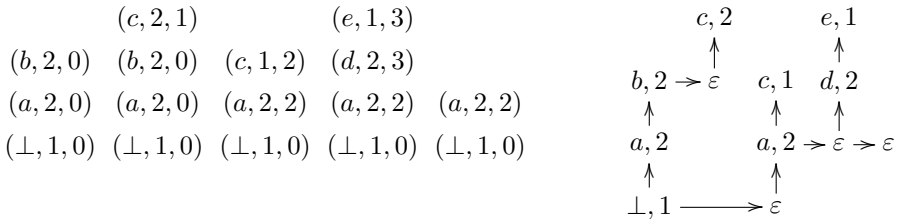


Figure 2 A stack s and its encoding $\mathrm{Enc}(s)$: right arrows lead to 1-successors (right successors), upward arrows lead to 0-successors (left successors).

fixed $(\sigma', l', k') \in \Sigma \times \{1,2\} \times \mathbb{N}$, and appropriate $w''_{j'} \in \Sigma^*$. For $\rho \in \big(\Sigma \times \{1,2\}\big) \cup \{\varepsilon\}$, we define recursively the $\big(\Sigma \times \{1,2\}\big) \cup \{\varepsilon\}$-labelled tree $\mathrm{Enc}(s, \rho)$ via

$$\mathrm{Enc}(s,\rho) := \begin{cases} \rho & \text{if } |w_1| = 1, n = 1 \\ \rho(\emptyset, \mathrm{Enc}({}_2 s_n, \varepsilon)) & \text{if } |w_1| = 1, n > 1 \\ \rho(\mathrm{Enc}({}_1 s'_n, (\sigma', l')), \emptyset) & \text{if } j = n, |w_1| > 1 \\ \rho(\mathrm{Enc}({}_1 s'_j, (\sigma', l')), \mathrm{Enc}({}_{j+1} s_n, \varepsilon)) & \text{otherwise.} \end{cases}$$

$\mathrm{Enc}(s) := \mathrm{Enc}(s, (\bot, 1))$ is called the (tree-)encoding of the stack $s \in \mathrm{Stck}(\Sigma)$.

Figure 2 shows a configuration and its encoding.

▶ Remark. In this encoding, the first block of a (σ, l, k)-blockline is encoded in a subtree whose root d is labelled (σ, l). For every node labelled by some element in $\Sigma \times \{1,2\}$, i.e., for every $d \in \mathrm{Enc}(s) \cap \{0,1\}^*0$, we can restore k from the position of d in $\mathrm{Enc}(s)$ as follows. If $l = 1$ then $k = |d|_0$, i.e., the number of occurrences of 0 in d. This is due to the fact that level 1 links always point to the preceding letter and that we always introduce a left-successor tree in order to encode letters that are higher in the stack. If $l = 2$ then

$$k = |\{d' \in \mathrm{dom}(\mathrm{Enc}(s)) \cap \{0,1\}^*1 : d' \leq_{\mathrm{lex}} d\}|,$$

where $\leq_{\mathrm{lex}}$ is the lexicographic order. This is due to the fact that every right-successor corresponds to the separation of some block from another block further left.

Having defined the encoding of stacks, we define the encoding of configurations.

▶ **Definition 11.** For $q \in Q$ and s some stack, we define $\mathrm{CEnc}(q, s) := q(\mathrm{Enc}(s), \emptyset)$.

The image of CEnc contains only trees of a very specific type. We call this class $\mathbb{T}_{\mathrm{Enc}}$.

▶ **Definition 12.** Let $\mathbb{T}_{\mathrm{Enc}}$ be the class of all trees T that satisfy the following conditions.

1. The root of T is labelled by some element of Q ($T(\varepsilon) \in Q$).
2. Every element of the form $\{0,1\}^*0$ is labelled by some $(\sigma, l) \in \Sigma \times \{1,2\}$; especially, $T(0) = (\bot, 1)$ and there are no other occurrences of $(\bot, 1)$ or $(\bot, 2)$.
3. Every element of the form $\{0,1\}^*1$ is labelled by ε.
4. $1 \notin \mathrm{dom}(T)$, $0 \in \mathrm{dom}(T)$.
5. For all $t \in T$, if $T(t0) = (\sigma, 1)$ then $T(t10) \neq (\sigma, 1)$.

▶ **Lemma 13** ([8]). *The image of* CEnc *is* $\mathbb{T}_{\mathrm{Enc}}$.

The following lemma shows that $\mathbb{T}_{\mathrm{Enc}}$ is a regular set.

▶ **Lemma 14.** *There is a finite automaton $\mathcal{A}_{\mathbb{T}_{\mathrm{Enc}}}$ with $f_0(\Sigma) := 2 + 3|\Sigma|$ many states that recognises $\mathbb{T}_{\mathrm{Enc}}$.*

Proof. Set $\mathcal{A}_{\mathbb{T}_{\mathrm{Enc}}} := (Q \cup (\Sigma \times \{1,2\}) \cup \{\varepsilon\}, Q_{\mathcal{A}}, \Delta_{\mathcal{A}}, q_I)$ where $Q_{\mathcal{A}}$ and $\Delta_{\mathcal{A}}$ are defined as follows. Let $Q_{\mathcal{A}} := \{\bot, q_I\} \cup (\Sigma \times \{1,2\}) \cup \{P_\sigma : \sigma \in \Sigma\}$. The states of the form (σ, i) are used to guess that a node of the tree is labelled by (σ, i) while the states P_σ are used to prohibit that the left successor of a certain node is labelled by $(\sigma, 1)$. The transitions ensure that whenever we guess that $d0$ is labelled by $(\sigma, 1)$ then $d1$ is reached in state P_σ ensuring that $d10$ cannot be labelled by $(\sigma, 1)$. $\Delta_{\mathcal{A}}$ is defined as follows.

- $(q_I, q, (\bot, 1), \bot) \in \Delta_{\mathcal{A}}$ for all $q \in Q$,
- $((\sigma, i), (\sigma, i), (\tau, 1), P_\tau) \in \Delta_{\mathcal{A}}$ for all $\sigma \in \Sigma$, $\tau \in \Sigma \setminus \{\bot\}, i \in \{1,2\}$,
- $((\sigma, i), (\sigma, i), (\tau, 2), P_\bot) \in \Delta_{\mathcal{A}}$ for all $\sigma \in \Sigma$, $\tau \in \Sigma \setminus \{\bot\}$, and $i \in \{1,2\}$,
- $((\sigma, i), (\sigma, i), (\tau, j), \bot) \in \Delta_{\mathcal{A}}$ for all $\sigma \in \Sigma$, $\tau \in \Sigma \setminus \{\bot\}, i, j \in \{1,2\}$,
- $((\sigma, i), (\sigma, i), \bot, P_\bot) \in \Delta_{\mathcal{A}}$ for all $\sigma \in \Sigma$, and $i \in \{1,2\}$,
- $((\sigma, i), (\sigma, i), \bot, \bot) \in \Delta_{\mathcal{A}}$ for all $\sigma \in \Sigma$, and $i \in \{1,2\}$,
- $(P_\sigma, \varepsilon, (\tau, 1), P_\tau) \in \Delta_{\mathcal{A}}$ for all $\sigma \in \Sigma$ and $\tau \in \Sigma \setminus \{\sigma, \bot\}$
- $(P_\sigma, \varepsilon, (\tau, 2), P_\bot) \in \Delta_{\mathcal{A}}$ for all $\sigma \in \Sigma$ and $\tau \in \Sigma \setminus \{\bot\}$
- $(P_\sigma, \varepsilon, (\tau, i), \bot) \in \Delta_{\mathcal{A}}$ for all $\sigma \in \Sigma$ and $(\tau, i) \in (\Sigma \times \{1,2\}) \setminus \{(\sigma, 1), (\bot, 1), (\bot, 2)\}$,
- $(P_\sigma, \varepsilon, \bot, P_\bot) \in \Delta_{\mathcal{A}}$ for all $\sigma \in \Sigma$
- $(P_\sigma, \varepsilon, \bot, \bot) \in \Delta_{\mathcal{A}}$ for all $\sigma \in \Sigma$ ◀

The next lemma is a straightforward observation concerning the depth of the encoding of a configuration in terms of the width and height of the stack.

▶ **Lemma 15.** *Given a stack s and a state q, such that (q, s) is reachable from the initial configuration by some path of length l then* $\mathrm{dp}(\mathrm{CEnc}(q, s)) < \mathrm{hgt}(s) + \mathrm{wdt}(s) \leq l + 2$.

Proof. We have seen that successors to the right correspond to the separation of different words of s. More precisely, $\mathrm{wdt}(s) = |\{d \in \{0,1\}^*\{1\} : d \in \mathrm{CEnc}(q, s)\}| + 1$. Furthermore, we have seen that every element $d \in \mathrm{CEnc}(q, s)$ encodes some word of length $|d|_0$. Thus, $\mathrm{hgt}(s) = \max\{|d|_0 : d \in \mathrm{CEnc}(q, s)\}$. We immediately conclude that $|d| < \mathrm{hgt}(s) + \mathrm{wdt}(s)$ for all $d \in \mathrm{CEnc}(q, s)$.

The second part of the claim is proved by induction. Note that the initial configuration is encoded by a tree of depth 2. Any stack operation increases the width or height of the stack by at most 1 and no operation increases the height and the width at the same time. ◀

3.4 Milestones

We now recall the concept of milestones from [8]. The milestones of a stack s are those substacks that every run to s has to pass. Thus, the concept of a milestone forms an essential key to understanding our pumping arguments in the next section.

▶ **Definition 16** (Milestone)**.** A substack s' of $s = w_1 : w_2 : \ldots : w_n$ is a *milestone* if $s' = w_1 : w_2 : \ldots : w_i : w'$ such that $0 \leq i < n$ and $w_i \sqcap w_{i+1} \leq w' \leq w_{i+1}$.[3] We denote by $\mathrm{MS}(s)$ the set of milestones of s.

▶ **Lemma 17** ([8])**.** *If s, t, m are stacks with $m \in \mathrm{MS}(t)$ but $m \not\leq s$, then every run from s to t visits m.*

[3] $\sqcap$ is the greatest common prefix operator.

▶ **Corollary 18.** *If ρ is a run from a stack s to a stack t, then ρ visits some stack $m \in \mathrm{MS}(t)$ with $\mathrm{dp}(m) \leq \mathrm{dp}(s) + 1$.*

Proof. ρ has to pass some common substack u of s and t. It is an easy exercise to show that pop_1 and pop_2-operations do not increase the depth of the encoding of a stack. If $u \in \mathrm{MS}(t)$ we are done. Otherwise, there is a minimal sequence of level 1 push operations that generate a milestone $m \in \mathrm{MS}(t)$ from u. For $c := \mathrm{wdt}(u)$, $\mathrm{top}_2(u) < w_{c-1} \sqcap w_c$ where w_i denotes the i-th word of t and $\mathrm{top}_2(m) = w_{c-1} \sqcap w_c$. Since w_{c-1} is also the $(c-1)$-st word of s, the height of m is bounded by $\mathrm{hgt}(s)$. It is straightforward to show that the encodings of m and u differ in exactly two nodes. There is some $d \in \mathrm{dom}(\mathrm{Enc}(u))$ such that $\mathrm{Enc}(m)$ is $\mathrm{Enc}(u)$ where we delete the node $d1$ and add some node $d01^{k_1}01^{k_2}\ldots01^{k_l+1}$ such that $d01^{k_1}01^{k_2}\ldots01^{k_l} \in \mathrm{dom}(\mathrm{Enc}(u))$. This operation increases the depth by at most 1. ◀

Milestones form an effectively regular set. This stems from the close correspondence between milestones of a stack s and the elements of $\mathrm{CEnc}(s)$ as follows.

▶ **Definition 19.** Let $c = (q, s)$ be some configuration. For $T := \mathrm{CEnc}(c)$ the encoding of c, let $d \in T \setminus \{\varepsilon\}$. Then the *left and downward closed tree induced by d* is $LT(d, T) := T{\restriction_D}$ where $D := \{d' \in T : d' \leq_{lex} d\} \setminus \{\varepsilon\}$. Then we denote by $\mathrm{LStck}(d, T)$ the unique stack s' such that $\mathrm{CEnc}(q, s') = LT(d, T)$. We call $\mathrm{LStck}(d, T)$ the *left stack induced by d*.

▶ Remark. $\mathrm{LStck}(d, \mathrm{CEnc}(q, s))$ is a substack of s for all $d \in \mathrm{dom}(\mathrm{Enc}(s))$. This observation follows from the fact that the left stack is induced by a lexicographically downward closed subset. In fact, $\mathrm{LStck}(d, \mathrm{Enc}(q, s))$ is a milestone of s. See [8] for more details.

▶ **Lemma 20.** *[8] The map given by $g : d \mapsto \mathrm{LStck}(d, \mathrm{CEnc}(q, s))$ is an order isomorphism between $(dom(\mathrm{CEnc}(q, s)) \setminus \{\varepsilon\}, \leq_{lex})$ and $(\mathrm{MS}(s), \leq)$.*

▶ **Lemma 21.** *There is an automaton $\mathcal{A}$ with 5 states such that for all configurations (q, s) and (q', m) the automaton $\mathcal{A}$ accepts $\mathrm{CEnc}(q', m) \otimes \mathrm{CEnc}(q, s)$ if and only if $m \in \mathrm{MS}(s)$.*

Proof. $\mathcal{A}$ has to check that $\mathrm{CEnc}(q', m)$ is a left and downward closed subtree of $\mathrm{CEnc}(q, s)$ (except for the label at the root). The states of $\mathcal{A}$ are $\{q_I, \bot, =, \neq, =^*\}$. The transition relation Δ consists of the following transitions:

1. $(q_I, (q_1, q_2), =, \bot)$ for $q_1, q_2 \in Q$
2. $(=, (X, X), =^*, =)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
3. $(=, (X, X), =, \bot)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
4. $(=, (X, X), \neq, \bot)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
5. $(=, (X, X), =, \neq)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
6. $(=, (X, X), \bot, =)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
7. $(=, (X, X), \bot, \neq)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
8. $(=, (X, X), \neq, \neq)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
9. $(=, (X, X), \bot, \bot)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
10. $(=^*, (X, X), =^*, =^*)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
11. $(=^*, (X, X), =^*, \bot)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
12. $(=^*, (X, X), \bot, =^*)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
13. $(=^*, (X, X), \bot, \bot)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
14. $(\neq, (\bot, X), \bot, \bot)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
15. $(\neq, (\bot, X), \neq, \bot)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
16. $(\neq, (\bot, X), \bot, \neq)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$,
17. $(\neq, (\bot, X), \neq, \neq)$ for $X \in \{\varepsilon\} \cup (\Sigma \times \{1, 2\})$.

◀

4 Pumping on Encodings of Configurations

The main result of [8] is that there is an automaton that accepts the encoding of a configuration if and only if this configuration is reachable from the initial one. This automaton guesses this run by labelling each node of the encoding with the initial and final state corresponding to the subrun starting at the corresponding milestone. In the following, we will use variants of this automaton in order to develop a pumping lemma on collapsible pushdown systems.

▶ **Theorem 22** ([8]). *For each collapsible pushdown system $\mathcal{S} = (\Sigma, Q, \Delta, q_0)$, there is a finite tree automaton $\mathcal{A}$ with*

$$f_1(Q, \Sigma) := 2 \cdot (2^{|Q \times Q|})^2 \cdot |Q|^2 \cdot |\Sigma \times \{1, 2\}| \cdot |\Sigma|$$

many states that accepts a tree $\mathrm{CEnc}(q, s)$ if and only if $(q, s) \in \mathrm{CPG}(\mathcal{S})$, i.e., if there is a run of $\mathcal{S}$ from the initial configuration to (q, s).

▶ Remark. In [8], we did non state the explicit bound on the number of states. This bound is extracted as follows. $\mathcal{A}$ guesses at each node $d \in \mathrm{CEnc}(q, s)$ states q_1, q_2 such that there is a run from $(q_1, \mathrm{LStck}(d, \mathrm{CEnc}(q, s)))$ to some configuration (q_2, t) where the definition of t depends on whether d is in the rightmost branch.

Let t' be the block encoded in the subtree rooted at d. If d is in the rightmost path, then $t = \mathrm{LStck}(d, \mathrm{CEnc}(q, s)) \setminus t'$.[4] At all other positions $t = (\mathrm{LStck}(d, \mathrm{CEnc}(q, s)) \setminus t') : \mathrm{top}_2(\mathrm{LStck}(d, \mathrm{CEnc}(q, s)))$. Thus, we obtain a factor 2 for keeping track of the rightmost branch. Furthermore, in order to verify the guesses q_1, q_2 at each node d, $\mathcal{A}$ stores the values of $\mathrm{Sym}(\mathrm{LStck}(d, \mathrm{CEnc}(q, s)))$, $\mathrm{CLvl}(\mathrm{LStck}(d, \mathrm{CEnc}(q, s)))$, $\mathrm{Sym}(\mathrm{pop}_1(\mathrm{LStck}(d, \mathrm{CEnc}(q, s))))$ and the existence of *loops* and *returns* starting at $\mathrm{LStck}(d, \mathrm{CEnc}(q, s))$. A loop starting in a stack s is a run from (q_1, s) to (q_2, s) for $q_1, q_2 \in Q$ that does not visit substacks of $\mathrm{pop}_2(s)$ and a return is a run from (q_1, s) to $(q_2, \mathrm{pop}_2(s))$ that does not visit any substack of $\mathrm{pop}_2(s)$ before its final configuration. At each node d, the automaton $\mathcal{A}$ has to keep track of the sets

$$\{(q_1, q_2) : \exists \text{ a loop from } (q_1, t) \text{ to } (q_2, t)\} \text{ and}$$
$$\{(q_1, q_2) : \exists \text{ a return from } (q_1, t) \text{ to } (q_2, \mathrm{pop}_2(t))\}$$

where $t = \mathrm{LStck}(d, \mathrm{CEnc}(q, s))$.

▶ **Corollary 23.** *For each collapsible pushdown system $\mathcal{S} = (\Sigma, Q, \Delta, q_0)$, there is a finite tree automaton $\mathcal{A}_\mathcal{S}$ such that $\mathcal{A}_\mathcal{S}$ accepts a tree T if and only if there is some configuration c such that $T = \mathrm{CEnc}(c)$ and c is contained in $\mathrm{CPG}(\mathcal{S})$. Moreover, $\mathcal{A}_\mathcal{S}$ has $f_2(\Sigma, Q) := f_0(\Sigma) \cdot f_1(Q, \Sigma)$ many states.*

Proof. $\mathcal{A}$ is the product of the automaton from Theorem 22 and that from Lemma 14. ◀

In fact, every run from the initial configuration to some configuration c induces an accepting run of $\mathcal{A}$ on $\mathrm{CEnc}(c)$. There is a close correspondence between the states of the run of $\mathcal{A}$ at positions $d \in \mathrm{CEnc}(c)$ and the states in which the run to c visits $\mathrm{LStck}(d, \mathrm{CEnc}(c))$. We state this correspondence in the following lemma.

▶ **Lemma 24** ([8]). *For each stack t and each state $q \in Q$ there is a state $q_{q,t}$ of $\mathcal{A}_\mathcal{S}$ such that the following holds. Given an arbitrary configuration $c = (q, s)$ such that $t \in \mathrm{MS}(s)$,*

[4] $s_1 \setminus s_2$ is an abbreviation for $\mathrm{pop}_2(s_1) : (\mathrm{top}_2(s_1) \setminus s_2)$.

there is a run ρ from the initial configuration to c that passes t in state q for the last time if and only if there is an accepting run $\rho_{\mathcal{A}_\mathcal{S}}$ of $\mathcal{A}_\mathcal{S}$ on $\mathrm{CEnc}(c)$ *such that $\rho_{\mathcal{A}_\mathcal{S}}(d) = q_{q,t}$ for d the unique node such that $t = \mathrm{LStck}(d, \mathrm{CEnc}(c))$.*

In analogy to the general pumping lemma 5, we now prove a specific pumping lemma for collapsible pushdown graphs. In order to obtain better bounds, we do not use the automaticity of the reachability predicate. Instead, we use reachability only restricted to pairs $(q_1, s_1), (q_2, s_2)$ where s_1 is a milestone of s_2.

▶ **Theorem 25.** *Let $\mathcal{S} = (\Sigma, Q, \Delta, q_0)$ be a* CPS*. Let ρ_0 be a run from the initial configuration to some configuration c with $\mathrm{len}(\rho_0) = l$. If ρ is a run starting at c of length*

$$\mathrm{len}(\rho) > f_3(Q, \Sigma, l) := |Q| \cdot (2|\Sigma| + 1)^{2^{l+2+f_2(Q,\Sigma)}}$$

then there are infinitely many runs starting at c.

Proof. If there are $i < j \leq \mathrm{len}(\rho)$ such that $\rho(i) = \rho(j)$ then $\pi_i := \rho{\restriction_{[0,i]}} \circ (\rho{\restriction_{[i,j]}})^i \circ \rho{\restriction_{[j,\mathrm{len}(\rho)]}}$ is an infinite sequence of runs starting at c.

Otherwise, the run visits more configurations than there are configurations whose encoding has depth at most $l + 2 + f_2(Q, \Sigma)$. Thus, there is some $i \leq \mathrm{len}(\rho)$ such that

$$\mathrm{dp}(\mathrm{CEnc}(\rho(i))) > l + 2 + f_2(Q, \Sigma)$$

Set $(q, s) := \rho(i)$. Due to Lemma 15, $\mathrm{dp}(\mathrm{CEnc}(c)) < l + 2$. Due to Corollary 18, there is a maximal $0 \leq j < i$ such that $\rho(j) = (\hat{q}, m)$ for some milestone $m \in \mathrm{MS}(s)$ and some state $\hat{q} \in Q$ such that $\mathrm{dp}(\mathrm{CEnc}(\hat{q}, m)) \leq l + 2$. Since m is a milestone of s, there is some node $d_m \in \mathrm{CEnc}(q, s)$ such that $\mathrm{LStck}(d_m, \mathrm{CEnc}(q, s)) = m$. This implies that the left and downward closed subtree induced by d_m is $\mathrm{Enc}(m)$.

Using Lemma 24, $\hat{q}$ and m define a state $q_{\hat{q},m}$ of $\mathcal{A}$ such that there is an accepting run of $\mathcal{A}$ on $\mathrm{CEnc}(q, s)$ that labels d_m with $q_{\hat{q},m}$.

Note that $LT(d_m, \mathrm{CEnc}(q, s))$ is a tree of depth at most $l + 2$. Hence, $\mathrm{CEnc}(q, s)$ contain a subtree of depth greater than $f_2(Q, \Sigma)$ that does not intersect with $LT(d_m, \mathrm{CEnc}(q, s))$. Since $f_2(Q, \Sigma)$ is a bound on the number of states of $\mathcal{A}$, Lemma 4 gives an infinite set of configurations $(q, s_1), (q, s_2), \ldots, (q, s_i), \ldots$ that are accepted by $\mathcal{A}$. Since the pumping does not affect $LT(d_m, \mathrm{CEnc}(q, s))$, we have $LT(d_m, \mathrm{CEnc}(q, s)) = LT(d_m, \mathrm{CEnc}(q, s_1)) = \cdots = LT(d_m, \mathrm{CEnc}(q, s_i)) = \ldots$ and the accepting run of $\mathcal{A}$ on $\mathrm{CEnc}(q, s_i)$ labels d by $q_{\hat{q},m}$. Using Lemma 24, we conclude that for each $1 \leq i$ there is a run π_i from the initial configuration to (q, s_i) passing $(\hat{q}, m)$ at position k_i. Recall that $\rho(j) = (\hat{q}, m)$. Thus, the composition $\rho{\restriction_{[0,j]}} \circ \pi_i{\restriction_{[k_i,\mathrm{len}(\pi_i)]}}$ is a run from c to (q, s_i). Hence, we constructed infinitely many runs starting at configuration c. ◀

5 ε-Contractions of Collapsible Pushdown Graphs of Level 2

In this section we lift the pumping lemma from the previous section to ε-contractions of collapsible pushdown systems. Let $\mathfrak{G}$ be a graph with labelled edges where the labels come from the set $\Gamma \cup \{\varepsilon\}$. The ε-contraction of $\mathfrak{G}$ is then the graph $\mathfrak{G}/\varepsilon$ that consists of the vertices of $\mathfrak{G}$ where v and v' are connected by a γ-labelled edge (for $\gamma \in \Gamma$) if there is a path from v to v' in $\mathfrak{G}$ such that all edges of this path are labelled by ε except for the last edge, which is labelled by γ. We denote by $\vdash^\gamma$ the relation induced by the γ-labelled edges in the ε-contraction. From now on, we consider the transitions of a collapsible pushdown system to be labelled with elements from some finite set $\Gamma \cup \{\varepsilon\}$.

We first prove a slight variation of Theorem 22. Then we show that in every finitely branching ε-contraction of some CPS the stack size of connected configurations cannot differ too much. Finally, we develop the analogue of Theorem 25 for ε-contractions of CPG.

▶ **Lemma 26.** *For each collapsible pushdown system $\mathcal{S} = (\Sigma, Q, \Delta, q_0)$ and each subset $\Delta' \subseteq \Delta$, there is a finite tree automaton $\mathcal{A}^{\Delta'}$ with $f_4 := 2 \cdot f_1(Q, \Sigma)$ many states that accepts a tree $\mathrm{CEnc}(q', t) \otimes \mathrm{CEnc}(q, s)$ for $t \in \mathrm{MS}(s)$ if and only if there is a run ρ from the initial configuration to (q, s) passing t at position i for the last time such that $\rho(i) = (q', t)$ and $\rho{\restriction}_{[i,\mathrm{len}(\rho)]}$ only uses transitions from Δ'.*

Proof. The automaton nondeterministically guesses the rightmost path of $\mathrm{CEnc}(q', t)$. After this path, i.e. on $\mathrm{dom}(\mathrm{CEnc}(q, s)) \setminus \mathrm{dom}(\mathrm{CEnc}(q', t))$ and at the lexicographically greatest node of $\mathrm{dom}(\mathrm{CEnc}(q', t))$, it simulates the automaton $\mathcal{A}$ from Theorem 22 but with respect to the transition relation Δ'. Along the rightmost path of $\mathrm{CEnc}(q', t)$ (except for the maximal node of this path), it simulates $\mathcal{A}$ in guessing final states for the corresponding subtrees. But it keeps the initial state fixed to q'. Thus, the automaton looks for runs to (q, s) that only use transitions from Δ', but it is forced to pass t' in state q'. ◀

▶ **Corollary 27.** *For each collapsible pushdown system $\mathcal{S} = (\Sigma, Q, \Delta, q_0)$ and each subset $\Delta' \subseteq \Delta$, there is a finite tree automaton $\mathcal{A}$ with*

$$f_5(Q, \Sigma) := 5 \cdot f_0(\Sigma) \cdot f_0(\Sigma) \cdot f_4(Q, \Sigma)$$

many states that accepts a tree T if and only if $T = \mathrm{CEnc}(q', t) \otimes \mathrm{CEnc}(q, s)$ for some configurations $(q', t), (q, s)$ such that $t \in \mathrm{MS}(s)$ and such that there is a run ρ from the initial configuration to (q, s) passing t at position i for the last time such that $\rho(i) = (q', t)$ and $\rho{\restriction}_{[i,\mathrm{len}(\rho)]}$ only uses transitions from Δ'.

Proof. T has to consists of two components, each one from $\mathbb{T}_{\mathrm{Enc}}$. Taking a product of two adaptations of the Automaton from Lemma 14 we can check that $T = \mathrm{CEnc}(q', t) \otimes \mathrm{CEnc}(q, s)$ for some configurations (q', t) and (q, s). Furthermore, taking the product with the automaton from Lemma 21, we can ensure that $t \in \mathrm{MS}(s)$. Finally, taking the product with the automaton from Lemma 26 yields the automaton $\mathcal{A}$. ◀

Completely analogously to the proof of Theorem 25 we now derive a bound of the size of stacks of configurations connected by an edge in a finitely branching ε-contraction of collapsible pushdown graphs.

▶ **Lemma 28.** *For each transition $\delta \in \Delta$ there is an automaton $\mathcal{A}_\delta$ with 10 states that accepts $\mathrm{CEnc}(q, s) \otimes \mathrm{CEnc}(q', s')$ if and only if $(q, s) \vdash^\delta (q', s')$.*

The proof of this lemma can be found in the long version of this article. It is obtained by explicit construction of the automaton informally described in [8].

▶ **Corollary 29.** *For each collapsible pushdown system $\mathcal{S} = (\Sigma, Q, \Delta, q_0)$ and each transition $\delta \in \Delta$, there is a finite tree automaton with $f_7(\Sigma) := 10 f_0(\Sigma) \cdot f_0(\Sigma)$ states that accepts a tree T, if and only if $T = \mathrm{CEnc}(q, s) \otimes \mathrm{CEnc}(q', s')$ and $(q, s) \vdash^\delta (q', s')$.*

Proof. T has to consists of two components, each one from $\mathbb{T}_{\mathrm{Enc}}$. Taking a product of two adaptations of the Automaton from Lemma 14 (one for each component) and of $\mathcal{A}_\delta$ from the previous lemma does the job. ◀

We now give a bound on the branching degree of ε-contractions of collapsible pushdown graphs.

▶ **Lemma 30.** *Let $\mathcal{S}$ be some collapsible pushdown system with stack alphabet Σ and state set Q. Set $\mathfrak{G} := \mathrm{CPG}(\mathcal{S})/\varepsilon$. If there are configurations $c', c \in \mathfrak{G}$ such that $\mathfrak{G} \models c' \vdash^\gamma c$ and*

$$\mathrm{dp}(\mathrm{CEnc}(c)) > 1 + \mathrm{dp}(\mathrm{CEnc}(c')) + f_7(\Sigma) \cdot f_5(Q, \Sigma)$$

then $\mathfrak{G}$ is infinitely branching.

Proof. Assume that $c' \vdash^\gamma c$ in $\mathfrak{G}$. Let $\Delta' \subseteq \Delta$ be the ε-labelled transitions of the CPS $\mathcal{S}$ generating $\mathfrak{G}$. Let $\mathcal{A}'$ denote the automaton of Corollary 27 with respect to $\mathcal{S}$ and Δ'. Furthermore, let $\mathcal{A}_\gamma$ denote the automaton of Corollary 29. There is an automaton $\hat{\mathcal{A}}$ that accepts a tree T if and only if $T = \mathrm{CEnc}(q_1, s_1) \otimes \mathrm{CEnc}(q_2, s_2) \otimes \mathrm{CEnc}(q_3, s_3)$ such that

1. $s_1 \in \mathrm{MS}(s_2)$,
2. $(q_1, s_1) \in \mathrm{CPG}(\mathcal{S})$,
3. there is a run from (q_1, s_1) to (q_2, s_2) that only uses transitions from Δ', and
4. $(q_2, s_2) \vdash^\gamma (q_3, s_3)$.

$\mathcal{A}$ is basically a product of $\mathcal{A}'$ and $\mathcal{A}_\gamma$. Thus, this automaton can be realised with $f_7(\Sigma) \cdot f_5(Q, \Sigma)$ many states.

Fix a run ρ witnessing that $c' \vdash^\gamma c$ in $\mathfrak{G}$. Writing $(q, s) := c$ and $\hat{c} := \rho(\mathrm{len}(\rho) - 1)$, Corollary 18 gives us a milestone m such that for some $q_m \in Q$ the automaton $\mathcal{A}$ accepts $\mathrm{CEnc}(q_m, m) \otimes \mathrm{CEnc}(\hat{c}) \otimes \mathrm{CEnc}(c)$. Furthermore,

$$\mathrm{dp}(\mathrm{CEnc}(c)) > \mathrm{dp}(\mathrm{CEnc}(q_m, m)) + f_7(\Sigma) \cdot f_5(Q, \Sigma).$$

Thus, we can apply the regular pumping argument to some subtree of $\mathrm{CEnc}(q_m, m) \otimes \mathrm{CEnc}(\hat{c}) \otimes \mathrm{CEnc}(c)$ where the first component is undefined. This yields infinitely many configurations $c_1, c_2, c_3, \dots$ such that $(q_m, m) \vdash^\gamma c_j$ for each $j \in \mathbb{N}$. Since $\rho{\restriction_{[0,i]}}$ is an ε-path from c' to (q_m, m), this implies $c' \vdash^\gamma c_j$ for each $j \in \mathbb{N}$ in $\mathfrak{G}$. Hence, $\vdash^\gamma$ in $\mathfrak{G}$ is infinitely branching at c'. ◀

▶ **Corollary 31.** *Let $\mathfrak{G}$ be the ε-contraction of some collapsible pushdown system with stack alphabet Σ and state set Q. If $\mathfrak{G}$ is finitely branching and if a path of length n connects the initial configuration with $c \in \mathfrak{G}$, then*

$$\mathrm{dp}(\mathrm{CEnc}(c)) \leq 2 + n\,(1 + f_7(\Sigma) \cdot f_5(Q, \Sigma))$$

The proof is by induction using the previous lemma. The straightforward adaptation of Theorem 25 yields the following pumping lemma for ε-contractions of collapsible pushdown graphs of level 2.

▶ **Theorem 32.** *Let $\mathcal{S} = (\Sigma, Q, \Delta, q_0)$ be a* CPS*. Let $\mathfrak{G} := \mathrm{CPG}(\mathcal{S})/\varepsilon$ be finitely branching. Let ρ_0 be a path from the initial configuration to some configuration c of length l in $\mathfrak{G}$.*

If there is a path ρ starting in c such that

$$\mathrm{len}(\rho) > f_6(Q, \Sigma, l) := |Q| \cdot (2|\Sigma| + 1)^{2^{L+K}}$$

for $L := 2 + l\,(1 + f_7(\Sigma) \cdot f_5(Q, \Sigma))$

and for $K := 1 + f_7(\Sigma) \cdot f_5(Q, \Sigma)$

then there are infinitely many paths in $\mathfrak{G}$ starting at c.

Proof. There may be $i < j$ such that $\rho(i) = \rho(j)$ and we can iterate $\rho{\restriction_{[i,j]}}$ arbitrarily many times. Otherwise, due to the length of ρ, there is some i such that

$$\mathrm{dp}(\mathrm{CEnc}(\rho(i))) > L + K \geq \mathrm{dp}(\mathrm{CEnc}(\rho(0))) + K.$$

Analogous to the proof of Lemma 30, pumping yields runs $\rho_1, \rho_2, \dots$ starting at c and ending in pairwise different configurations of $\mathfrak{G}$. ◀

Using the same bounds, the second general pumping lemma 6 has a collapsible pushdown version:

▶ **Theorem 33.** *Let $\mathcal{S} = (\Sigma, Q, \Delta, q_0)$ be a* CPS*. Let $\mathfrak{G} := \mathrm{CPG}(\mathcal{S})/\varepsilon$ be finitely branching. Let ρ_0 be a path from the initial configuration to some configuration c of length l in $\mathfrak{G}$.*

If there are more than $f_6(Q, \Sigma, l)$ many configurations reachable from c then there are infinitely many paths in $\mathfrak{G}$ starting at c.

Again, one of the configurations reachable form c must be encoded by a tree of depth $\mathrm{dp}(\mathrm{CEnc}(\rho(i))) > \mathrm{dp}(\mathrm{CEnc}(\rho(0))) + K$ and we may apply the pumping argument from the previous proof.

▶ Remark. There is some function f_8 such that $f_6(Q, \Sigma, l) \leq 2^{2^{f_8(Q,\Sigma)\cdot l}}$.

6 Applications

▶ **Corollary 34.** *Let $\mathcal{S}$ be some* CPS *and $\mathcal{G} := \mathrm{CPG}(\mathcal{S})/\varepsilon$. It is decidable whether $\mathfrak{G}$ is finite.*

We can also use the pumping lemma in order to prove that certain graphs are not ε-contractions of CPG.

▶ **Example 35.** Let $\varphi : \mathbb{N} \to \mathbb{N}$ be an unbounded monotone function. The tree

$$\mathcal{T} = \left\{0^{i-1}1^j \in \{0,1\}^* : j \leq 2^{2^{\varphi(i)\cdot i}} + 1\right\}$$

(with left and right successor relation) is not the ε-contraction of any CPG of level 2.

Heading for a contradiction, assume there was such a CPS $\mathcal{S}$. Choose $k \in \mathbb{N}$ such that $\varphi(k) \geq f_8(Q, \Sigma)$. Thus, $2^{2^{\varphi(k)\cdot k}} \geq f_6(Q, \Sigma, k)$ whence we may apply Theorem 32 to the path connecting $0^{k-1}1$ with $0^{k-1}1^{\varphi(k)k+1}$ and obtain infinitely many paths starting in $0^{k-1}1$. But this contradicts the definition of $\mathcal{T}$.

Using the second pumping lemma, one proves analogously that the tree

$$\mathcal{T} = \{0^n : n \in \mathbb{N}\} \cup \{0^n 1 : n \in \mathbb{N}\} \cup \left\{0^{i-1}1j : j \leq 2^{2^{\varphi(i)\cdot i}} + 1\right\}$$

is not the ε-contraction of any CPS of level 2.

7 Conclusions

To our knowledge, we presented the first pumping lemma for collapsible pushdown graphs of level 2. Moreover, the result also improves Hayashi's pumping lemma for indexed languages [7]. An analysis of his proof shows that his pumping lemma applies to runs of level 2 pushdown systems that have length three-fold exponential in the size of the pushdown system. Our lemma already applies to runs that have length doubly exponential in the size of the system.

Unfortunately, our approach does not extend directly to higher levels of the collapsible pushdown hierarchy. Higher levels are not tree-automatic. But perhaps it is possible to represent the reachability relations of higher-order collapsible pushdown graphs by other types of automata for which pumping lemmas exist. These could then be turned into pumping lemmas for higher-order collapsible pushdown graphs. Another approach towards pumping on higher-order graphs stems from the the technical tools of milestones and loops developed in [8]. It is an interesting question whether these notions can be adapted to higher levels in order to obtain pumping lemmas for all higher-order (collapsible) pushdown graphs.

Acknowledgments

I thank the anonymous referees for their very useful comments.

References

1 A. Blumensath. On the structure of graphs in the Caucal hierarchy. *Theoretical Computer Science*, 400:19–45, 2008.
2 C. H. Broadbent. Private communication. September 2010.
3 A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2003*, volume 2914 of *LNCS*, pages 112–123. Springer, 2003.
4 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 2007. release October, 12th 2007.
5 Robert H. Gilman. A shrinking lemma for indexed languages. *Theor. Comput. Sci.*, 163(1&2):277–281, 1996.
6 M. Hague, A. S. Murawski, C-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS '08: Proceedings of the 2008 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 452–461, 2008.
7 Takeshi Hayashi. On derivation trees of indexed grammars. *Publ. RIMS, Kyoto Univ.*, 9:61–92, 1973.
8 A. Kartzow. Collapsible pushdown graphs of level 2 are tree-automatic. In *STACS 10*, volume 5 of *LIPIcs*, pages 501–512. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.
9 A. Kartzow. *First-Order Model Checking On Generalisations of Pushdown Graphs*. PhD thesis, Technische Universität Darmstadt, 2011. Unpublished, submitted in December 2010.
10 T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FOSSACS'02*, volume 2303 of *LNCS*, pages 205–222. Springer, 2002.
11 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. *SIGPLAN Not.*, 44:416–428, January 2009.
12 Pawel Parys. The pumping lemma is incorrect? unpublished, June 2010.
13 Pawel Parys. Collapse Operation Increases Expressive Power of Deterministic Higher Order Pushdown Automata. In Thomas Schwentick and Christoph Dürr, editors, *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 603–614, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Decidability Issues for Two-Variable Logics with Several Linear Orders *

Emanuel Kieroński

Institute of Computer Science, University of Wrocław, Poland
kiero@cs.uni.wroc.pl

Abstract

We show that the satisfiability and the finite satisfiability problems for two-variable logic, FO^2, over the class of structures with three linear orders, are undecidable. This sharpens an earlier result that FO^2 with eight linear orders is undecidable. The theorem holds for a restricted case in which linear orders are the only non-unary relations. Recently, a contrasting result has been shown, that the finite satisfiability problem for FO^2 with two linear orders and with no additional non-unary relations is decidable. We observe that our proof can be adapted to some interesting fragments of FO^2, in particular it works for the two-variable guarded fragment, GF^2, even if the order relations are used only as guards. Finally, we show that GF^2 with an arbitrary number of linear orders which can be used only as guards becomes decidable if except linear orders only unary relations are allowed.

1998 ACM Subject Classification F.4 Mathematical Logic and Formal Languages

Keywords and phrases Two-variable logic, linear orders, guarded fragment, decidability

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.337

1 Introduction

In the field of logic in computer science the two-variable fragment of first order logic, FO^2, plays a prominent role. With respect to the number of variables it appears to be the maximal fragment whose satisfiability problem is decidable. The importance of FO^2 can be justified by the fact that it, or its natural extensions and variants, embeds many formalisms used in computer science, such as modal, temporal or description logics.

The decidability of FO^2 was shown in [18] by establishing a finite model property, namely, that every satisfiable formula has a finite model of size at most doubly exponential with respect to its length. This bound on the size of models was later improved in [7] to singly exponential, which implied NExpTime-upper bound on the complexity of the satisfiability problem. A corresponding lower bound follows from [15, 5], so the satisfiability problem for FO^2 is NExpTime-complete.

One particular drawback of FO^2 is that it cannot express transitivity of a binary relation. Similarly, it is not possible to say that a relation is, e.g., an equivalence relation or a linear order. Such properties of relations are very natural and desirable in practical applications. Thus researchers started to investigate FO^2 over restricted classes of structures, in which some distinguished binary symbols have to be interpreted as transitive relations, equivalences, or linear orders. The idea for such a kind of research comes from the world of modal logics, where, e.g., in Kripke structures for multimodal logic K4 accessibility relations are transitive and for multimodal logic S5 they are equivalences. Linear orders are very natural

* Partially supported by Polish Ministry of Science and Higher Education grant N N206 371339.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 337–351

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

when we consider temporal logics, where they model time flow, but can be also applicable in different scenarios, like in databases or description logics, to compare objects with respect to some parameters.

Unfortunately, the results are generally negative. It appeared that both the satisfiability and the finite satisfiability problems for FO^2 are undecidable in the presence of several equivalence or several transitive relations [8, 9]. These results were later strengthened: FO^2 is undecidable in the presence of two transitive relations [11, 10], three equivalence relations [12], one transitive and one equivalence relation [14], or eight linear orders [19]. On the positive side it is known that FO^2 with one or two equivalence relations [12, 14], or with one linear order [19] are decidable.

A related line of work, motivated by XML, concerns the so called *data words*. A data word is a word over a finite alphabet. Positions of a word are naturally ordered by the linear order and may be related by an equivalence relation (such an equivalence relation models equality of data values). It was shown in [3] that FO^2 is decidable over data words, even in the case when except the linear order we are allowed to use the associated successor relation. Some other interesting results related to data words have been recently obtained in [4], [17] and [20]. In particular it is shown in [17] that FO^2 is decidable over words whose positions are ordered by two linear orders, with the assumption that the orders are only accessible by the successor relations.

In this paper we perform a next step towards completing the classification of FO^2 with linear orders. We show that the satisfiability and the finite satisfiability problems for FO^2 are undecidable in the presence of three linear orders. The proof works for a restricted language, in which, besides three linear orders, only unary predicates are used. This theorem improves the above mentioned result from [19], where eight linear orders were used. It also sharpens a theorem from [21] that FO^2 is undecidable in the presence of two linear orders and one total preorder. Our result seems to be optimal with respect to the number of linear orders, since it contrasts with the main theorem from [21], that the finite satisfiability problem for FO^2 with a linear order and a total preorder (and thus also for FO^2 with two linear orders) is decidable. The proof of the last result works only in the case in which the order relations are the only non-unary symbols; it is very likely however that it can be extended to the general case.

It is an interesting question if there exists a natural decidable fragment of FO^2 in which elements could be compared by an unbounded number of linear orders (or, at least, by more than two orders). When looking for analogous fragments with transitive or equivalence relations the attention is often turned to the two-variable *guarded fragment*, GF^2. In the guarded fragment each occurrence of a quantifier has to be relativised by an atomic formula containing all the variables that are free in the scope of this quantifier, e.g. $\forall xy(x < y \rightarrow (Px \wedge Qy \wedge Bxy))$. The guarded fragment was introduced in [1] to simulate the way accessibility relations in modal logics or roles in description logics are used. The satisfiability problem for the guarded fragment is 2ExpTime-complete and for its two-variable version – ExpTime-complete [6].

It appeared that GF^2 is decidable with an arbitrary number of transitive or equivalence relations, if the usage of transitive or equivalence symbols is restricted only to guards [22, 11, 13]. This last restriction is natural, since to the obtained fragment we may still translate multimodal logics K5, S4 or some description logics with transitive roles.

In the case of linear orders the situation appears to be different. Our undecidability proof for FO^2 can be easily adapted to the case of GF^2, even if linear orders are allowed to appear only as guards. This can be done by enforcing that some additional binary relations

are identical to the linear orders. On the other hand, if we assume that linear orders are the only non-unary symbols and are used only as guards then GF^2 becomes decidable. The obtained variant allows only for a very limited interaction among different linear orders (in fact, because of the syntactic restrictions, such interaction can be obtained only in an indirect way), however it seems that not much more can be done: we explain that, e.g., extending the fragment by allowing guards built from conjunctions of atoms instead of just atoms, e.g. guards like $x \leq_1 y \wedge x \leq_2 y \wedge y \leq_3 x$, leads to undecidability.

The organisation of the paper is as follows. In Section 2 we present our main undecidability result for FO^2 with three linear orders and discuss some of its refinements. In Section 3 we show that GF^2 with an arbitrary number of linear orders is decidable if linear orders are used only as guards and if there are no additional non-unary symbols.

2 Undecidability

2.1 Tilings and grids

The reduction of the tiling problem to satisfiability of some variants of two-variable logic was presented in [8, 9]. Some ramifications, particularly suited for the case of linear orders, were given in [19]. For convenience we present (adaptations of) some basic definitions and lemmas (without proofs) from [19].

Let $\mathfrak{G}_{\mathbb{Z}}$ be the canonical grid structure on $\mathbb{Z} \times \mathbb{Z}$: $\mathfrak{G}_{\mathbb{Z}} = (\mathbb{Z}^2, H, V)$, $H = \{((p,q),(p+1,q)) \colon p,q \in \mathbb{Z}\}$, $V = \{((p,q),(p,q+1)) \colon p,q \in \mathbb{Z}\}$. Similarly, let $\mathfrak{G}_{\mathbb{N}}$ be the canonical grid on $\mathbb{N} \times \mathbb{N}$ and let $\mathfrak{G}_m$ denote the standard grid on a finite $m \times m$ torus: $\mathfrak{G}_m = (\mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/m\mathbb{Z}, H, V)$, $H = \{((p,q),(p',q)) \colon p' - p \equiv 1 \mod m\}$, $V = \{((p,q),(p,q')) \colon q' - q \equiv 1 \mod m\}$.

Let $\mathfrak{G}_i = (G_i, H_i, V_i), i = 1, 2$. $\mathfrak{G}_1$ is *homomorphically embeddable* into $\mathfrak{G}_2$ if there is a homomorphism $\pi : \mathfrak{G}_1 \to \mathfrak{G}_2$, i.e. a mapping π such that for all $v, v' \in G_1$: $(v,v') \in H_1 \Rightarrow (\pi(v), \pi(v')) \in H_2$ and $(v,v') \in V_1 \Rightarrow (\pi(v), \pi(v')) \in V_2$.

We are interested in structures which are grid-like in the following sense.

▶ **Definition 1.** An infinite structure $\mathfrak{G} = (G, H, V)$ is called *grid-like* if $\mathfrak{G}_{\mathbb{N}}$ is homomorphically embeddable into $\mathfrak{G}$; a finite $\mathfrak{G}$ is grid-like if some $\mathfrak{G}_m$ is homomorphically embeddable into $\mathfrak{G}$.

Grid-likeness is implied by a simple local criterion. We say that H is *complete over* V in $\mathfrak{G} = (G, H, V)$ if $\mathfrak{G}$ satisfies $\forall xyx'y'((Hxy \wedge Vxx' \wedge Vyy') \to Hx'y')$.

▶ **Lemma 2.** *Assume that $\mathfrak{G} = (G, H, V)$ satisfies the FO^2-axiom $\forall x(\exists y Hxy \wedge \exists y Vxy)$. If H is complete over V, then $\mathfrak{G}$ is grid-like.*

▶ **Lemma 3.** *Let $\mathcal{C}$ be a class of structures. If there exists an FO^2 sentence η such that:*

(a) *$\mathfrak{G}_{\mathbb{Z}}$ can be expanded to a structure in $\mathcal{C}$ satisfying η,*
(b) *for every $n \in \mathbb{N}$ there exists $k \in \mathbb{N}$ such that $\mathfrak{G}_{kn}$ can be expanded to a structure in $\mathcal{C}$ satisfying η,*
(c) *every model of η from $\mathcal{C}$ is grid-like,*

then both satisfiability and finite satisfiability of FO^2 over $\mathcal{C}$ are undecidable. In fact FO^2 forms even a conservative reduction class over $\mathcal{C}$. If at least (a) and (c) hold then the (general) satisfiability problem is undecidable.

For a more detailed exposition of the technique see [19]. For some background on conservative reduction classes see [2].

The general idea of our proof of the undecidability of FO^2 with three linear orders is similar to the idea from the proof for the case of eight linear orders from [19], however details are much tricker.

To postpone some technical problems and present the main ideas of the proof clearly, in the first instance we consider only the (general) satisfiability case. First, we describe the expansion $\bar{\mathfrak{G}}_{\mathbb{Z}}$ of the standard infinite $\mathbb{Z} \times \mathbb{Z}$ grid by three linear orders $\leq_1, \leq_2, \leq_3$ and some unary predicates. Then we construct a formula η capturing some important properties of $\bar{\mathfrak{G}}_{\mathbb{Z}}$. We argue that every model of η, interpreting symbols $\leq_i$ as linear orders, is grid-like. By Lemma 3 this implies the undecidability of the satisfiability problem for FO^2 over the class of structures with three linear orders.

Further, we describe expansions of the finite $12k \times 12k$ grids, $\bar{\mathfrak{G}}'_{12k}$ in a signature containing some additionall unary symbols. We modify slightly the formula η, obtaining η' which will be satisfied in $\bar{\mathfrak{G}}'_{12k}$ for all $k \in \mathbb{N}$. It will also appear that η' satisfies all assumptions of Lemma 3, which shows that FO^2 forms a conservative reduction class (in particular the finite satisfiability problem is undecidable) over the class of structures with three linear orders.

2.2 Intended infinite model

We describe the expansion $\bar{\mathfrak{G}}_{\mathbb{Z}}$ of the standard $\mathbb{Z} \times \mathbb{Z}$ grid. The basic repeating pattern of the grid expansion consists of 24 elements, forming a 4×6 rectangle. To distinguish types of elements inside such rectangles we use unary predicates P_{ij}, $0 \leq i \leq 3$, $0 \leq j \leq 5$. Namely, if $a = (k, l)$ then $\bar{\mathfrak{G}}_{\mathbb{Z}} \models P_{ij}a$ if and only if $i = k \mod 4$ and $j = l \mod 6$.

In Fig. 1 we illustrate the order $\leq_1$. The set of elements $\mathbb{Z} \times \mathbb{Z}$ is divided into horizontal $\leq_1$-*zones*, each of them consisting of three rows of elements. Formally, the $\leq_1$-zones are defined as $\mathbb{Z}_k^{\leq_1} = \{(i, 3k), (i, 3k+1), (i, 3k+2) : i \in \mathbb{Z}\}$ for $k \in \mathbb{Z}$. If $a \in Z_k^{\leq_1}$, $b \in Z_l^{\leq_1}$, and $k < l$ then $\bar{\mathfrak{G}}_{\mathbb{Z}} \models b \leq_1 a$. The points in a zone are organised in U-shaped six-element blocks, called $\leq_1$-*blocks*. If for elements $a, b \in Z_k^{\leq_1}$, a belongs to a $\leq_1$-block located to the left from the $\leq_1$-block of b then $\bar{\mathfrak{G}}_{\mathbb{Z}} \models b \leq_1 a$. Look at Fig. 1 to see the $\leq_1$-ordering inside the $\leq_1$-blocks. Note that the $\leq_1$-blocks in the odd zones are shifted by 1 with respect to the even zones.

The orders $\leq_2$ and $\leq_3$ follow the same pattern, but are shifted with respect to the order $\leq_1$. To obtain the picture for $\leq_2$ we shift the picture for $\leq_1$ by the vector $(1, 1)$. Similarly, the picture for $\leq_3$ is obtained by shifting the picture for $\leq_1$ by $(0, 2)$. This implies that the zones determined by different order relations do not coincide. See Fig. 2 to see how $\leq_i$-blocks of all three orders are located in the grid. For clarity $\leq_i$-relations are shown only inside $\leq_i$-blocks. Recall that $\leq_i$-arrows among $\leq_i$-zones go from up to down and among the $\leq_i$-blocks inside a zone – from right to left.

Fig. 3 shows relations $\leq_1$ and $\leq_3$ between the neighbouring points from two consecutive rows of the grid. Note that elements connected by H or by V are related by $\leq_1$, $\leq_3$ incompatibly. This observation extends to the crucial property of $\bar{\mathfrak{G}}_{\mathbb{Z}}$, which will be used to axiomatise the grid relations H, V: all three orders coincide on points whose y-coordinates differ by at least 3, or which belong to the same row and their x-coordinates differ by at least 2; and, on the other hand, points connected by the grid relation H or by the grid relation V are related incompatibly by some pair of the orders. We state it precisely in the following observation. We also formalise another important property of our grid expansion (part (iv)) which will be captured by the formula η; this will allow to show that in all models of η, H is complete over V.

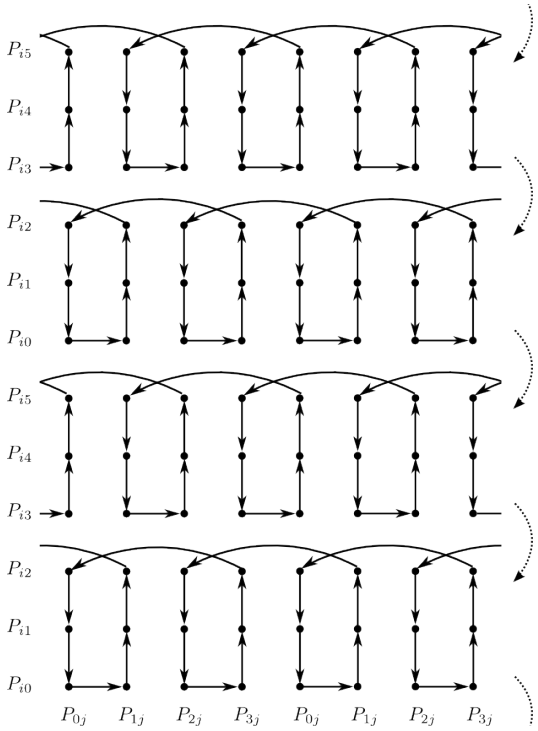


Figure 1 The order $\leq_1$. Solid arrows represent the successor relation, dotted arrows illustrate relations among $\leq_1$-zones. The lower-left element is the point $(0,0)$.

▶ Observation 4. (i) Let $(k,l),(k',l') \in \mathbb{Z}\times\mathbb{Z}$ be two points in $\bar{\mathfrak{G}}_{\mathbb{Z}}$, such that $l'-l \geq 3$. Then for all i we have $(k',l') \leq_i (k,l)$.
(ii) Let $(k,l),(k',l') \in \mathbb{Z}\times\mathbb{Z}$ be two points in $\bar{\mathfrak{G}}_{\mathbb{Z}}$, such that $l=l'$ and $k'-k \geq 2$. Then for all i we have $(k',l') \leq_i (k,l)$.
(iii) If (k,l) and (k',l') are connected by H or by V, i.e, if $k=k'$ and $l'-l=1$ or $l=l'$ and $k'-k=1$, then there exist i,j such that $(k,l) \leq_i (k',l')$ and $(k',l') \leq_j (k,l)$ or $(k,l) \leq_j (k',l')$ and $(k',l') \leq_i (k,l)$. Namely, if $l \mod 3 = 0$ then $i=1, j=3$, if $l \mod 3 = 1$ then $i=1, j=2$, and if $l \mod 3 = 2$ then $i=2, j=3$.
(iv) For all points $a,b,c,d \in \mathbb{Z}\times\mathbb{Z}$, if $\bar{\mathfrak{G}}_{\mathbb{Z}} \models Vba \wedge Hbc \wedge Vcd$, then there exist i,j such that $a \leq_i b \leq_i c \leq_i d$ and $d \leq_j c \leq_j b \leq_j a$. Namely, if $b=(k,l)$ then i,j can be chosen as in point (iii).

Proof. Claim (i) follows from the fact that for all i the point (k',l') belongs to a $\leq_i$-zone located above the zone of (k,l). Claim (ii) follows from the fact that for all i both points belong to the same $\leq_i$-zone, and that for all orders the point (k,l) belongs to a $\leq_i$-block located to the left from the $\leq_i$-block of (k',l'). Claims (iii),(iv) follow from an inspection of Fig. 2. ◀

2.3 The formula η

The formula η consists of four conjuncts $\eta = \eta_G \wedge \eta_H \wedge \eta_V \wedge \eta_C$. The first conjunct explicitly enforces horizontal and vertical successors in the grid:

$$\eta_G = \forall x(\exists y Hxy \wedge \exists y Vxy).$$

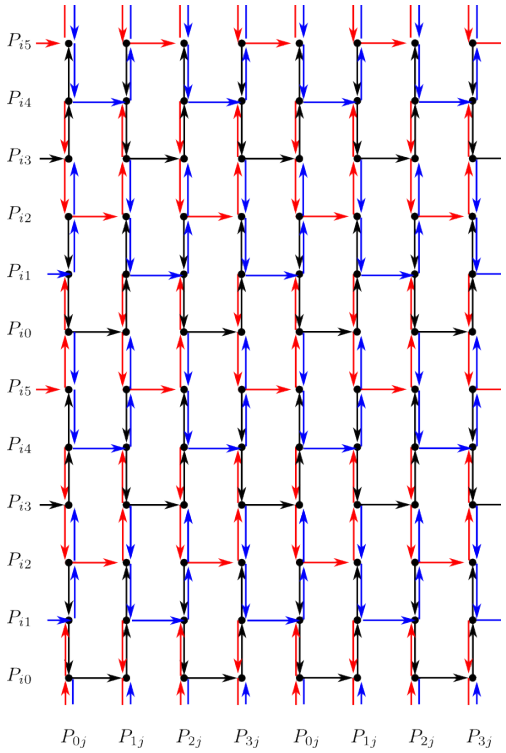


Figure 2 U-shaped blocks in the orders: $\leq_1$ (black arrows), $\leq_2$ (blue arrows), $\leq_3$ (red arrows). For clarity only successor $\leq_i$-connections inside $\leq_i$-blocks are shown.

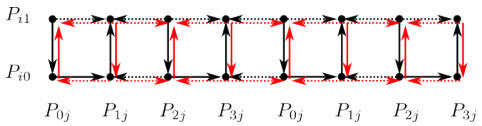


Figure 3 Relations $\leq_1$ (black arrows) and $\leq_3$ (red arrows) between the neighbouring points from two consecutive rows. Solid arrows represent successor relations, dotted arrows represent non-successor $\leq_i$-relations.

The next conjunct η_H axiomatises H:

$$\eta_H = \forall xy(Hxy \leftrightarrow \bigvee_{\substack{0 \leq i \leq 3 \\ 0 \leq j \leq 5}} (P_{ij}x \wedge P_{i+1,j}y \wedge \lambda^H_{ij}(x,y))),$$

where $i+1$ is calculated modulo 4, and $\lambda^H_{ij}(x,y)$ says how points x, y are related by two of the three orders; namely λ^H_{i0} and λ^H_{i3} speak about $\leq_1$ and $\leq_3$, λ^H_{i1} and λ^H_{i4} speak about $\leq_1$ and $\leq_2$, λ^H_{i2} and λ^H_{i5} speak about $\leq_2$ and $\leq_3$, e.g.:

$$\lambda^H_{00} = x \leq_1 y \wedge y \leq_3 x,$$

$$\lambda^H_{01} = x \leq_1 y \wedge y \leq_2 x,$$

$$\lambda^H_{31} = y \leq_1 x \wedge x \leq_2 y.$$

The next conjunct η_V speaks about V-connections. It is similar to η_H, however this time we impose only the implication from left to right:

$$\eta_V = \forall xy(Vxy \to \bigvee_{\substack{0 \le i \le 3 \\ 0 \le j \le 5}} (P_{ij}x \wedge P_{i,j+1}y \wedge \lambda^V_{ij}(x,y)),$$

where $j+1$ is calculated modulo 6; again λ^V_{i0} and λ^V_{i3} speak about $\le_1$ and $\le_3$, λ^V_{i1} and λ^V_{i4} speak about $\le_1$ and $\le_2$, λ^V_{i2} and λ^V_{i5} speak about $\le_2$ and $\le_3$, e.g.:

$$\lambda^V_{00} = y \le_1 x \wedge x \le_3 y,$$

$$\lambda^V_{10} = x \le_1 y \wedge y \le_3 x,$$

$$\lambda^V_{21} = y \le_1 x \wedge x \le_2 y.$$

Finally, η_C says that some points, related incompatibly by two of the three orders are connected by the third one in a specific way:

$$\eta_C = \forall xy \bigwedge_{\substack{0 \le i \le 3 \\ 0 \le j \le 5}} ((P_{ij}x \wedge P_{i+1,j}y) \to \kappa_{ij}(x,y)),$$

where $i+1$ is calculated modulo 4. Formulae κ_{i0} and κ_{i3} enforce $\le_1$ relations, κ_{i1} and κ_{i4} – $\le_2$ relations, κ_{i2} and κ_{i5} – $\le_3$ relations, e.g.:

$$\kappa_{01} = (x \le_1 y \wedge y \le_3 x) \to y \le_2 x$$

$$\kappa_{15} = (x \le_1 y \wedge y \le_2 x) \to x \le_3 y$$

It can be readily checked that $\bar{\mathfrak{G}}_{\mathbb{Z}}$ is a model of η. In particular, the implication from right to left in η_H does not impose any unwanted H-connections. Indeed, by Observation 4 (i), (ii) the formula $\bigvee_{i,j}(P_{ij}a \wedge P_{i+1,j}b \wedge \lambda^H_{ij}(a,b))$ is not satisfied by non-neighbouring points a, b of the grid, since $\lambda^H_{ij}(a,b)$ says that a, b are related incompatibly by some two orders.

2.4 Grid-likeness

Now let us see that every model $\mathfrak{M} \models \eta$ interpreting $\le_1, \le_2, \le_3$ as linear orders[1] is grid like. Since $\mathfrak{M} \models \eta_G$, by Lemma 2, it suffices to check that H is complete over V. Assume, that $a, b, c, d \in M$ are such that $\mathfrak{M} \models Hbc \wedge Vba \wedge Vcd$. We want to see that $\mathfrak{M} \models Had$. We need to consider several cases, depending on the P_{ij}-type of b. Let us go through one of them. Assume that $\mathfrak{M} \models P_{00}b$. Then, the implications from left to right in η_H and η_V imply

$$\mathfrak{M} \models P_{10}c \wedge P_{01}a \wedge P_{11}d \wedge a \le_1 b \wedge b \le_1 c \wedge c \le_1 d \wedge d \le_3 c \wedge c \le_3 b \wedge b \le_3 a.$$

Since $\le_1$ and $\le_3$ are linear orders, and thus transitive, it follows that $\mathfrak{M} \models a \le_1 d \wedge d \le_3 a$. Now, consider η_C. It follows that $\mathfrak{M} \models \kappa_{01}(a,d)$. The implication in κ_{01} guarantees that $\mathfrak{M} \models d \le_2 a$. Thus $\mathfrak{M} \models P_{01}a \wedge P_{11}d \wedge \lambda^H_{01}(a,d)$, so the implication from right to left in η_H finally enforces $\mathfrak{M} \models Had$.

The remaining cases can be treated in a similar way.

This finishes the proof of the undecidability of the general satisfiability problem. To obtain the undecidability of finite satisfiability we need to work further on some details.

[1] Actually, for grid-likeness it is enough to assume that they are interpreted as transitive relations.

2.5 Finite models

We describe first how to construct our intended expansions $\bar{\mathfrak{G}}'_{12k}$ of the standard grids on $12k \times 12k$ tori, for $k \geq 1$. Then we explain how to modify η to a formula η' which is satisfied in such expansions, without losing the property that all models of η' are grid-like.

To simplify the presentation we describe the grid expansion $\bar{\mathfrak{G}}'_{12}$. The larger grid expansions are constructed analogously. Let $\bar{\mathfrak{G}}_{12}$ be the restriction of $\bar{\mathfrak{G}}_{\mathbb{Z}}$ to the set $\{0, \ldots, 11\} \times \{0, \ldots, 11\}$ in which additionally, for all l, the element $(11, l)$ is connected by H to $(0, l)$, and the element $(l, 11)$ is connected by V to $(l, 0)$ (i.e. the sides of the square are appropriately glued). $\bar{\mathfrak{G}}_{12}$ is not a model of η for some trivial reasons; among other things the implication from left to right in η_H is violated, e.g. $(11, 1)$ is inappropriately related to $(0, 1)$ by $\leq_1$.

Thus we slightly modify $\bar{\mathfrak{G}}_{12}$ to obtain $\bar{\mathfrak{G}}'_{12}$. In Fig. 4 the order $\leq_1$ is shown. The $\leq_1$-zones are defined analogously to the infinite case. In all odd zones, i.e. zones built from elements of types P_{i3}, P_{i4}, P_{i5}, the $\leq_1$-connections remain as they are in $\bar{\mathfrak{G}}_{12}$. The $\leq_1$-connections are modified in even zones. We describe the zone built from the rows $0, 1, 2$. The element $(10, 2)$ is made the minimal element in this zone. The next elements in the order are $(10, 1)$, $(10, 0)$, as in $\bar{\mathfrak{G}}_{12}$, the next one however is not $(11, 0)$ but $(8, 2)$. Then the order coincides with the order in $\bar{\mathfrak{G}}_{12}$ until the element $(1, 2)$ is reached. Its successor is $(11, 0)$, the next element is $(11, 1)$, and the maximal element in this zone is $(11, 2)$. More intuitively, we may think that the rightmost U in $\bar{\mathfrak{G}}_{12}$ is cut into two parts: the left one is made minimal in the zone and the right one – maximal, with respect to $\leq_1$.

Analogously to the case of infinite models, to obtain the pictures for $\leq_2$ and $\leq_3$ we shift the picture for $\leq_1$ by the vectors $(1, 1)$ and $(0, 2)$, respectively, taking into account that this time shifts are made on a torus, so, e.g., the minimal element with respect to $\leq_2$ will be the element $(0, 0)$.

We also introduce new unary symbols $S_0 - S_3$, intended to mark four consecutive columns of the grid (columns $10, 11, 0, 1$ in our example) and $Z_0 - Z_3$, intended to mark four consecutive rows of the grid (rows $2, 1, 0, 11$ in our example). Their relevance will become clear in a moment.

The described structure $\bar{\mathfrak{G}}'_{12}$ satisfies η_G, η_V and the implication from left to right in η_H. Unfortunately, parts (i) and (ii) of Observation 4 are not true this time, which makes the implication from right to left in η_H not satisfied. Let us explain why.

Note first that η_H enforces H-connections between distant elements from the same row. Consider e.g. the element $(11, 1)$. In its row this element is maximal with respect to $\leq_1$ and minimal with respect to $\leq_2$. Thus η_H enforces a H-connection e.g. from $(2, 1)$ to $(11, 1)$.

Similarly, some unwanted H-connections are enforced also between elements from different zones. Each of the orders divides the set of elements into four zones. In Fig. 5 it is shown how $\leq_1$-, $\leq_2$- and $\leq_3$-zones are related by $\leq_1$, $\leq_2$ and $\leq_3$, respectively. Note that the elements in the row marked Z_1 belong to the $\leq_3$-zone which is minimal with respect to $\leq_3$, and to the $\leq_2$- and $\leq_3$-zones which are maximal with respect to, resp., $\leq_1$ and $\leq_3$. Similarly, the elements in the row marked Z_2 belong to the $\leq_2$- and $\leq_3$-zones which are minimal with respect to, resp., $\leq_2$ and $\leq_3$, and to the $\leq_1$-zone which is maximal with respect to $\leq_1$. This means that η_H enforces some unwanted H-connections to (or from) Z_1 and Z_2, from (or to) some distant elements in the grid, e.g. the element $(1, 0)$ should be connected by H to $(2, 6)$.

To fix the problems we use the mentioned unary relations S_0, S_1, S_2, S_3 and Z_0, Z_1, Z_2, Z_3. Let

$$\alpha^S(x, y) = (S_0x \wedge S_1y) \vee (S_1x \wedge S_2y) \vee (S_2x \wedge S_3y) \vee (\neg S_1x \wedge \neg S_1y \wedge \neg S_2x \wedge \neg S_2y),$$

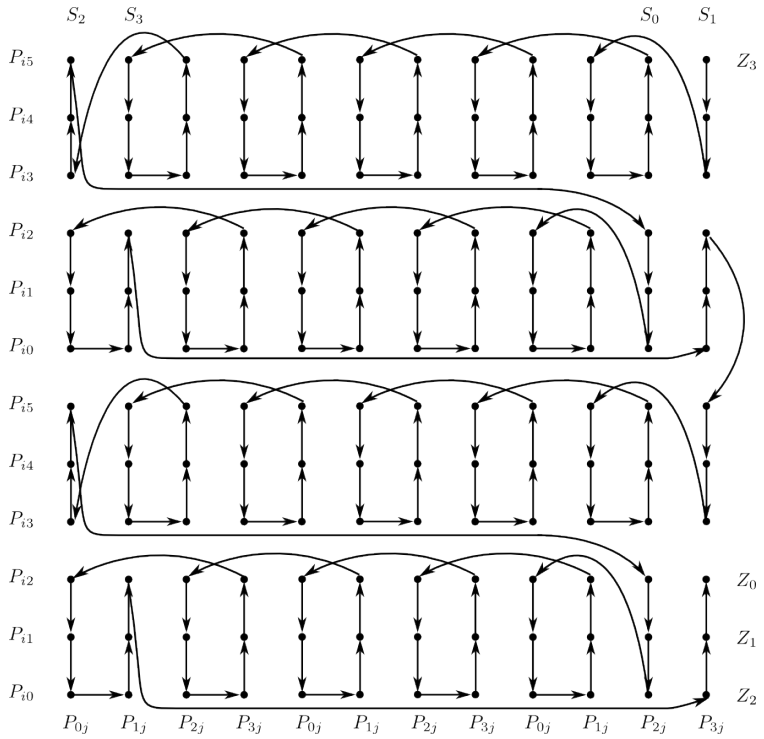


Figure 4 The order $\leq_1$ in the finite grid $\bar{\mathfrak{G}}'_{12}$

$$\alpha^Z(x,y) = \bigwedge_{0\leq i\leq 3} (Z_i x \leftrightarrow Z_i y).$$

For $\bar{\mathfrak{G}}'_{12k}$ we have now a slightly weaker observation than Observation 4 part (i) and (ii).

▶ Observation 5. Let $a=(k,l), b=(k',l') \in \mathbb{Z}\times\mathbb{Z}$ be two distinct points in $\bar{\mathfrak{G}}'_{12k}$, such that $\bar{\mathfrak{G}}'_{12k} \models \alpha^S(a,b) \wedge \alpha^Z(a,b)$. Then:

(i) If the distance in the torus between the row l and the row l' is at least 3 then for all i we have $(k',l') \leq_i (k,l)$ or for all i we have $(k,l) \leq_i (k',l')$.
(ii) If $l=l'$ and the distance in the torus between columns k' and k is at least 2 then for all i we have $(k',l') \leq_i (k,l)$ or for all i we have i we have $(k,l) \leq_i (k',l')$.

Proof. To see claim (i) note that elements a,b cannot belong to the rows marked by Z_i (since they satisfy $\alpha^Z(a,b)$). For the remaining rows an argument similar to the argument from the proof of Observation 4 (i) works. To see claim (ii) note that the elements a,b cannot belong to the columns marked S_1 or S_2 (since they satisfy $\alpha^S(a,b)$). For the remaining columns an argument similar to the argument from the proof of Observation 4 (ii) works. ◀

Observe that points (iii) and (iv) from Observation 4 remain true.

We modify η to allow and impose H-connection to S_1 only from S_0, to S_2 only form S_1, from S_1 only to S_2, from S_2 only to S_3; and to Z_i only from Z_i, for $0\leq i\leq 3$.

We change η_H to:

$$\eta'_H = \forall xy(Hxy \leftrightarrow (\alpha^S(x,y) \wedge \alpha^Z(x,y) \wedge \bigvee_{\substack{0\leq i\leq 3\\ 0\leq j\leq 5}} (P_{ij}x \wedge P_{i+1,j}y \wedge \lambda^H_{ij}(x,y)))).$$

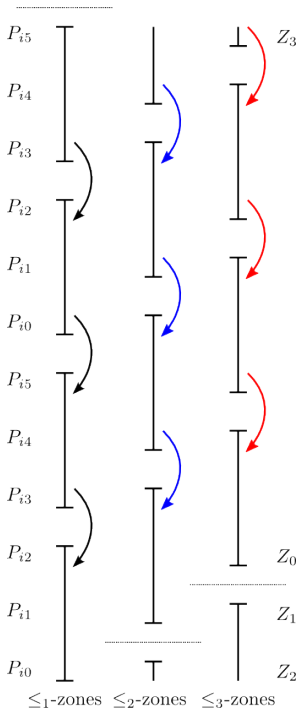


Figure 5 The $\leq_1$- $\leq_2$- and $\leq_3$-zones in $\bar{\mathfrak{G}}'_{12}$

We define additional conjuncts which say that V-connected elements are given consistent S_i- and Z_i-values:

$$\eta_S = \bigwedge_{0\leq i\leq 3} \forall xy(Vxy \to (S_i x \leftrightarrow S_i y)),$$

$$\eta_Z = \forall xy(Vxy \to ((\delta(x) \wedge \delta(y)) \vee (\delta(x) \wedge Z_3 y) \vee (Z_0 x \wedge \delta(y)) \vee \bigvee_{1\leq i\leq 3} (Z_i x \wedge Z_{i-1} y))),$$

where $\delta(x) = \bigwedge_{0\leq i\leq 3} \neg Z_i x$.

We modify also η_C, since it may also generate some unwanted relations:

$$\eta'_C = \forall xy \bigwedge_{\substack{0\,\leq\, i\,\leq\, 3\\ 0\,\leq\, j\,\leq\, 5}} (P_{ij}x \wedge P_{i+1,j}y \wedge \alpha^S(x,y) \wedge \alpha^Z(x,y)) \to \kappa_{ij}(x,y)).$$

Finally, let the conjunct η_U says that every element satisfies at most one of the S_i-predicates and at most one of the Z_i-predicates, and the S_i and Z_i-values imply proper P_{ij} values, e.g. $\forall x(Z_0 x \to (P_{02}x \vee P_{12}x \vee P_{22}x \vee P_{32}x))$.

Now let $\eta' = \eta_G \wedge \eta'_H \wedge \eta_V \wedge \eta_S \wedge \eta_Z \wedge \eta'_C \wedge \eta_U$. Every grid $\mathfrak{G}_{12k}$ can now be expanded to a model $\bar{\mathfrak{G}}'_{12k}$ of η' analogously to the described expansion of $\mathfrak{G}_{12}$. Also the infinite grid $\mathfrak{G}_{\mathbb{Z}}$ has an expansion to a model of η'. It is enough to take $\bar{\mathfrak{G}}_{\mathbb{Z}}$ and mark columns $-2, -1, 0, 1$ with, resp., S_0, S_1, S_2, S_3, and rows $2, 1, 0, -1$ with, resp., Z_0, Z_1, Z_2, Z_3.

Let us finally sketch a fragment of the argument that every model $\mathfrak{M}$ of η' interpreting $\leq_i$ as linear orders is grid-like. Assume, that $a, b, c, d \in M$ are such that $\mathfrak{M} \models Hbc \wedge Vba \wedge Vcd$. We want to see that $\mathfrak{M} \models Had$. This time the cases we have to consider are distinguished not only by the values o P_{ij} but also by the values of the additional relations S_i, Z_i. Let us

consider one of the cases, namely, $\mathfrak{M} \models P_{00}b \wedge S_2b \wedge Z_2b$. Then, the implication from left to right in η'_H, and the formulae η_V, η_S, and η_Z imply

$$\mathfrak{M} \models P_{10}c \wedge P_{01}a \wedge P_{11}d \wedge S_2a \wedge Z_1a \wedge S_3c \wedge Z_2c \wedge S_3d \wedge Z_1d$$

and

$$\mathfrak{M} \models a \leq_1 b \wedge b \leq_1 c \wedge c \leq_1 d \wedge d \leq_3 c \wedge c \leq_3 b \wedge b \leq_3 a.$$

Since $\leq_1$ and $\leq_3$ are linear orders, and thus transitive, it follows that $\mathfrak{M} \models a \leq_1 d \wedge d \leq_3 a$. Now, consider η'_C. Note that $\alpha^S(a,d)$ and $\alpha^Z(a,d)$ are true. It follows that $\mathfrak{M} \models \kappa_{01}(a,d)$. The implication in κ_{01} guarantees that $\mathfrak{M} \models d \leq_2 a$. Thus $\mathfrak{M} \models P_{01}a \wedge P_{11}d \wedge \lambda^H_{01}(a,d) \wedge \alpha^S(a,d) \wedge \alpha^Z(a,d)$. Finally, the implication from right to left in η_H enforces $\mathfrak{M} \models Had$.

We left the remaining cases to the reader.

We have proved that FO^2 forms a conservative reduction class over the class of structures with three linear orders.

2.6 Remarks on the proof and discussion

In our proof we use the binary symbols H and V. They are convenient to present the construction but do not play a crucial role. In a reduction from the tiling problem they can be simulated by combinations of unary predicates and the order relations. Namely, in η the conjunct η_G can be substituted by the conjunction of formulae enforcing for every x the existence of two elements related to x by the linear orders in a specific way: $\bigwedge_{i,j} \forall x(P_{ij}x \rightarrow \exists y(\eta^H_{ij}(x,y) \wedge P_{i+1,j}y) \wedge \exists y(\eta^V_{ij}(x,y) \wedge P_{i,j+1}y))$. The formulae η_H and η_V can then be omitted. Thus we obtain the following, strong version of our main undecidability result.

▶ **Theorem 6.** FO^2 *forms a conservative reduction class over the structures with three linear orders and no additional non-unary symbols.*

A question arises whether there exists an elegant and useful fragment of FO^2 which is decidable in the presence of an arbitrary number of linear orders (or at least in the presence of three linear orders). A natural candidate is the two-variable guarded fragment, GF^2. Let us recall the definition of the guarded fragment. The guarded fragment, GF, of first-order logic is defined as the least set of formulae such that: (i) every atomic formula belongs to GF; (ii) GF is closed under logical connectives $\neg, \vee, \wedge, \rightarrow$; and (iii) quantifiers are relativised by atoms, i.e. if $\varphi(\mathbf{x},\mathbf{y})$ is a formula of GF and $\gamma(\mathbf{x},\mathbf{y})$ is an atomic formula containing all the free variables of φ, then the formulae $\forall\mathbf{y}(\gamma(\mathbf{x},\mathbf{y}) \rightarrow \varphi(\mathbf{x},\mathbf{y}))$ and $\exists\mathbf{y}(\gamma(\mathbf{x},\mathbf{y}) \wedge \varphi(\mathbf{x},\mathbf{y}))$ belong to GF. The atoms $\gamma(\mathbf{x},\mathbf{y})$ are called *guards*.

Syntactically, not all of the formulae we use in our undecidability proof are guarded. However there is no problem to make them guarded, since linear orders are total and thus can be used as guards if necessary, e.g. $\forall xy\psi(x,y)$ can be rewritten as $\forall xy(x \leq_1 y \rightarrow \psi(x,y)) \wedge \forall xy(y \leq_1 x \rightarrow \psi(x,y))$. Thus GF^2 is undecidable in the presence of three linear orders. This situation is similar to the case of GF^2 with equivalence or transitive relations, which are also undecidable (with three equivalences [12], and with two transitive relations [11, 10]). However if we restrict the usage of special relations (i.e. equivalence or transitive relations) to guards only, then GF^2 becomes decidable, with an arbitrary number of special relations [22, 11, 13]. Unfortunately, a similar restriction does not help in the case of linear orders. A simple formula $\forall xy(x \leq_i y \rightarrow (x \neq y \rightarrow (R_ixy \wedge \neg R_iyx))) \wedge \forall xR_ixx$, in which R_i is a fresh binary symbol, enforces R_i to behave exactly as $\leq_i$. Thus R_i can replace all occurrences of $\leq_i$ outside guards.

▶ **Corollary 7.** *The satisfiability and the finite satisfiability problems for* GF^2*, in the class of structures with three linear orders, are undecidable even if linear orders are used only in guards.*

We emphasise that to obtain Corollary 7 some binary symbols except linear orders are required. It appears that if we allow only unary symbols except linear orders and allow to use linear orders only as guards then, as it is argued in the next section, GF^2 becomes decidable with an arbitrary number of linear orders. The obtained decidable variant, which will be called *monadic* GF^2, allows only for a very restricted interaction among different linear orders (in fact, because of the syntactic restrictions such interaction can be obtained only in an indirect way). However, it seems that the situation cannot be improved too much. For example, if instead of just linear orders we allow conjunctions of linear orders as guards then the logic becomes undecidable. This fact can be inferred using the exponential translation of FO^2 to a variant of Boolean modal logic from [16], but can be also proved directly, by observing that the formulae we construct in Section 2 can be rewritten to the desired variant. Indeed, consider the place which looks most problematically, i.e. the formula η_C. It says e.g. that elements x, y satisfying $P_{01}x$ and $P_{11}y$ which are related by $\leq_1$ and $\leq_3$ in the following way: $x \leq_1 y \wedge y \leq_3 x$ should satisfy also $y \leq_2 x$. This can be enforced by saying: $\forall xy((x \leq_1 y \wedge y \leq_3 x \wedge x \leq_2 y) \rightarrow (\neg P_{01}x \vee \neg P_{11}y))$. Again we use the fact that x, y has to be connected by $\leq_2$ and we only forbid the connection in the unwanted direction.

▶ **Corollary 8.** *The satisfiability and the finite satisfiability problems for the extension of monadic* GF^2 *with three linear orders, which allows conjunctions of atoms of the form* $x \leq_i y$ *and* $y \leq_i x$ *(for* $i = 1, 2, 3$*) as guards, are undecidable.*

3 Decidability

In this section we work with signatures of the form $(\sigma, \leq_1, \ldots, \leq_k)$, where σ is a set of unary symbols and $\leq_i$ are binary symbols. We assume that the equality is also allowed. Formally, monadic GF^2 is the fragment of GF^2 containing formulae over such signatures in which symbols $\leq_1, \ldots, \leq_k$ are used only as guards. We consider satisfiability of monadic GF^2 in the class of structures in which $\leq_1, \ldots, \leq_k$ are interpreted as linear orders, which we denote as $\mathcal{LIN}(\leq_1, \ldots, \leq_k)$. We will simply say that a monadic GF^2 sentence φ has a model (is satisfiable, finitely satisfiable) if it has a model (is satisfiable, finitely satisfiable) in $\mathcal{LIN}(\leq_1, \ldots, \leq_k)$.

A 1-*type* (over σ) is a subset of σ. If α is a 1-type then we denote by $\alpha(x)$ the conjunction of the atoms Px, for all $P \in \alpha$, and the atoms $\neg Qx$, for all $Q \notin \alpha$. For a given structure $\mathfrak{A}$ we say that an element a *realises* a type α if $\mathfrak{A} \models \alpha(a)$.

▶ **Definition 9.** A monadic GF^2 sentence φ is in *normal form* if it is a conjunction of formulae of the following form:

- $\exists x(\gamma(x) \wedge \psi(x))$,
- $\forall x(\gamma(x) \rightarrow \exists y(x \leq_i y \wedge x \neq y \wedge \psi(x, y)))$,
- $\forall x(\gamma(x) \rightarrow \exists y(y \leq_i x \wedge x \neq y \wedge \psi(x, y)))$,
- $\forall x(\gamma(x) \rightarrow \psi(x))$,
- $\forall xy(x \leq_i y \rightarrow (x \neq y \rightarrow \psi(x, y)))$.

where all $\gamma(x)$ are atomic formulae (possibly of the form $x = x$), and $\psi(x)$, $\psi(x, y)$ are quantifier-free formulae over monadic vocabulary σ.

The (finite) satisfiability problem for monadic GF^2 can be reduced to the (finite) satisfiability problem for disjunctions of exponential number of linearly bounded monadic GF^2

sentences in normal form. See [22] for the proof of a similar result. Since we are going to show that (finite) sastisfiability is in NExpTime it is enough to consider formulae in normal form.

The decidability proof for monadic GF^2 is based on the proof for FO^2 with one linear order from [19]. Roughly speaking, after fixing the universe, the (slightly simplified) construction from [19] is applied here to the particular orders. Below we present a sketch of the proof.

3.1 General satisfiability

▶ **Definition 10.** Let a tuple $(\mathcal{T}, \mathcal{K}, \mathcal{S}^1, \dots \mathcal{S}^k)$ be such that:

- $\mathcal{T}$ is a set of 1-types over σ,
- $\mathcal{K}$ is a subset of $\mathcal{T}$, called the set of *royal* 1-types,
- for every $1 \leq i \leq k$, $\mathcal{S}^i = (S^i_1, \dots, S^i_{k_i})$ is a sequence of subsets of $\mathcal{T}$, such that $\bigcup_{j=1}^{k_i} S^i_j = \mathcal{T}$, each type from $\mathcal{K}$ belongs to exactly one set from S^i, and the types from $\mathcal{K}$ appear only in singletons.

We say that such a tuple is a *certificate of satisfiability* for a normal form monadic GF^2 sentence φ if the following conditions hold:

(a) For every conjunct of φ of the form $\exists x(\gamma(x) \wedge \psi(x))$ there exists a type $\alpha \in \mathcal{T}$ such that $\alpha(x) \models \gamma(x) \wedge \psi(x)$.
(b) For every i, j, for every type $\alpha \in S^i_j$ and for every conjunct of φ of the form $\forall x(\gamma(x) \rightarrow \exists y(x \leq_i y \wedge x \neq y \wedge \psi(x,y)))$, if $\alpha(x) \models \gamma(x)$ then there exists α' in $S^i_{j'}$, such that $\alpha(x), \alpha'(y) \models \psi(x,y)$, where $j' \geq j$, and if $\alpha \in \mathcal{K}$ then $j' > j$.
(c) For every i, j, for every type $\alpha \in S^i_j$ and for every conjunct of φ of the form $\forall x(\gamma(x) \rightarrow \exists y(y \leq_i x \wedge x \neq y \wedge \psi(x,y)))$, if $\alpha(x) \models \gamma(x)$ then there exists α' in $S^i_{j'}$, such that $\alpha(x), \alpha'(y) \models \psi(x,y)$, where $j' \leq j$, and if $\alpha \in \mathcal{K}$ then $j' < j$.
(d) For every type $\alpha \in \mathcal{T}$ and for every conjunct of φ of the form $\forall x(\gamma(x) \rightarrow \psi(x))$, we have $\alpha(x) \models \gamma(x) \rightarrow \psi(x)$.
(e) For every i, $j \leq j'$, for every pair of types $\alpha \in S^i_j$, $\alpha' \in S^i_{j'}$, such that it is not the case that $\alpha = \alpha'$ and $\alpha \in \mathcal{K}$, then for every conjunct of the form $\forall xy(x \leq_i y \rightarrow (x \neq y \rightarrow \psi(x,y)))$, we have $\alpha(x), \alpha'(y) \models \psi(x,y)$.

▶ **Lemma 11.** *Let φ be a monadic GF^2 sentence in the normal form. Then φ is satisfiable if and only if it has a certificate of satisfiability.*

Proof. $\Leftarrow$ Assume that $(\mathcal{T}, \mathcal{K}, \mathcal{S}^1, \dots \mathcal{S}^k)$ is a certificate of satisfiability for φ. We build a model $\mathfrak{A}$ whose universe A consists of exactly one realisation of each type from $\mathcal{K}$ and infinitely many realisations of each type from $\mathcal{T} \setminus \mathcal{K}$. For every i we define the order $\leq_i$. We split A into sets $A^i_1, \dots, A^i_{k_i}$ in such a way that A^i_j contains infinitely many realisations of $\alpha \in S^i_j$ if $\alpha \notin \mathcal{K}$ and exactly one realisation of $\alpha \in S^i_j$ if $\alpha \in \mathcal{K}$. Now if $a \in A^i_j$, $a' \in A^i_{j'}$ and $j < j'$ then we set $\mathfrak{A} \models a \leq_i a'$. If S^i_j consists of non-royal types $\alpha^0, \dots, \alpha^{l-1}$ then we make the order $\leq_i$ on A^i_j isomorphic to the natural order on $\mathbb{Z}$, in such a way that the element corresponding to the number m has 1-type $\alpha^{m \mod l}$. It is readily checked that the conditions on the certificate imply that $\mathfrak{A} \models \varphi$.

$\Rightarrow$ Let $\mathfrak{A} \models \varphi$. We show how to extract a certificate of satisfiability for φ from $\mathfrak{A}$. We define $\mathcal{T}$ to be the set of 1-types realised in $\mathfrak{A}$ and $\mathcal{K}$ to be the set of 1-types realised exactly once in $\mathfrak{A}$.

For a given i and a type $\alpha \in \mathcal{T}$ we define B^i_α to be the minimal set containing all the realisations of α such that for all $a \leq_i b \leq_i c$, if $a, c \in B^i_\alpha$ then $b \in B^i_\alpha$. Let us denote by

$A^i_{\alpha^-}$ the subset of A consisting of the elements smaller than all the elements from B^i_α, and by $A^i_{\alpha^+}$ the union of B^i_α and $A^i_{\alpha^-}$. Observe that the sets $A^i_{\alpha^+}, A^i_{\alpha^-}$ are closed downwards. Let $F^i_0, \ldots, F^i_{k_i}$ be an ordered list of all the sets from $\{A^i_{\alpha^+}, A^i_{\alpha^-} : \alpha \in \mathcal{T}\}$ such that if $k < l$ then $F^i_k \subseteq F^i_l$. Note that k_i is linear in the number of 1-types and that $F^i_0 = \emptyset$ and $F^i_{k_i} = A$. Let us define $D^i_j = F^i_j \setminus F^i_{j-1}$ for $1 \leq j \leq k_i$. The sets D^i_j are called *i-regions*.

Let S^i_j be the set of 1-types realised in D^i_j. This finishes the definition of $(\mathcal{T}, \mathcal{K}, \mathcal{S}^1, \ldots \mathcal{S}^k)$.

The properties (a)-(d) of the certificate are satisfied in an obvious way. Let us prove (e). If $j < j'$ then the conclusion is straightforward. Similarly, if $j = j'$ and $\alpha = \alpha'$ is a royal type. Assume that $j = j'$ and $\alpha, \alpha' \in S^i_j$ are two non-royal types. It is enough to show that in $\mathfrak{A}$ there are two distinct elements a_α, b_α of type α and two distinct elements $a'_{\alpha'}, b'_{\alpha'}$ of type α' such that $\mathfrak{A} \models a_\alpha \leq_i a'_{\alpha'}$ and $\mathfrak{A} \models b'_{\alpha'} \leq_i b_\alpha$.

If $\alpha = \alpha'$ then we have at least two realisations a, b of α in $\mathfrak{A}$. Assume that $a \leq_i b$. Then we can take $a_\alpha = b'_\alpha = a$ and $a'_\alpha = b_\alpha = b$. If $\alpha \neq \alpha'$ consider elements a, a' from $\mathfrak{A}$, of types α, α', respectively, in the i-region D^i_j. Assume that $a \leq_i a'$ (the opposite case is symmetric). Assume that there is no realisation of α, which is greater, with respect to $\leq_i$ than a'. It means that $a' \notin B^i_\alpha$ (since otherwise B^i_α would not be minimal). Since $a \in B^i_\alpha$ we have a contradiction with the assumption that a, a' are member of the same i-region. So there exists a realisation b of α such that $\mathfrak{A} \models a' \leq_i b$. We can take $a_\alpha = a$, $b_\alpha = b$, $a_{\alpha'} = b_{\alpha'} = a'$. ◀

The construction in the proof of the lemma shows that every satisfiable monadic GF^2 sentence φ in the normal form has a certificate of size polynomial in the number of 1-types. Since we may assume that σ contains only symbols appearing in φ it implies that the size of a certificate can bounded exponentially in $|\varphi|$. Checking that a given tuple is indeed a certificate of satisfiability can easily be done in polynomial time. Thus:

▶ **Corollary 12.** *The satisfiability problem for monadic* GF^2 *is decidable in* NExpTime.

3.2 Finite satisfiability

The case of finite satisfiability is even simpler than the case of general satisfiability.

▶ **Lemma 13.** *Let φ be a* GF^2 *sentence in normal form. If φ is finitely satisfiable then φ has a model with at most $2k \cdot 2^{|\sigma|}$ elements, where k is the number of linear orders in the signature.*

Proof. Let $\mathfrak{A}$ be a model of φ. Mark in $\mathfrak{A}$ all the elements whose 1-types are realised only once. For every 1-type α, such that there are at least two realisations of α in $\mathfrak{A}$, and for every $0 \leq i \leq k$, mark the $\leq_i$-minimal and the $\leq_i$-maximal realisations of α. Let $\mathfrak{A}'$ be the substructure of $\mathfrak{A}$ induced by the marked elements. It is easy to verify that $\mathfrak{A}' \models \varphi$. ◀

▶ **Corollary 14.** *The finite satisfiability problem for monadic* GF^2 *is in* NExpTime.

3.3 Lower bound

The satisfiability problem for GF^2 in the class of all structures is in ExpTime [6]. On the other hand FO^2 is NExpTime-hard even if only unary symbols are allowed [15, 5]. Clearly such monadic FO^2 can be reduced to monadic GF^2 with just one linear order $\leq$, since this order can alway be used as a guard, e.g. a formula $\forall xy\psi(x, y)$ can be translated to $\forall xy(x \leq y \to \psi(x, y)) \wedge \forall xy(y \leq x \to \psi(x, y))$. Thus:

▶ **Theorem 15.** *The satisfiability and the finite satisfiability problems for monadic* GF^2 *are* NExpTime*-complete.*

References

1 H. Andréka, J. van Benthem and I. Németi, *Modal languages and bounded fragments of predicate logic*, Journal of Philosophical Logic, 27, 1998, 217-274.

2 E. Börger, E. Grädel and Y. Gurevich, *The Classical Decision Problem*, Springer, 1997.

3 M. Bojańczyk, C. David, A. Muscholl, T. Schwentick and L. Segoufin, *Two-variable logic on words with data*, Proc. of 21st IEEE Symp. on Logic in Computer Science, LICS 2006, 7-16. 82, 1996, 353-367.

4 C. David, L. Libkin and T. Tan, *On the satisfiability of two-variable logic over data words*, Proc. of 17th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2010, LNCS 6397, 248-262.

5 M. Fürer, *The computentional complexity of the unconstrained limited domino problem (with implications for logical decision problems)*. Logical Machines: Decision Problems and Complexity, LNCS 171, 1981, 312-319.

6 E. Grädel, *On the restraining power of guards*, J. of Symb. Logic, 64, 1999, 1719-1742.

7 E. Grädel, P. Kolaitis and M. Vardi, *On the decision problem for two-variable first order logic*, Bulletin of Symbolic Logic, 3, 1997, 53-96.

8 E. Grädel and M. Otto, *On logics with two variables*, Theoretical Computer Science, 224, 1999, 77-113.

9 E. Grädel, M. Otto and E. Rosen, *Undecidability results on two-variable first-order logic*, Archive of Mathematical Logic, 38, 1999, 313-354.

10 Y. Kazakov, *Saturation-based decision procedures for extensions of the guarded fragment*, PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 2006.

11 E. Kieroński, *Results on the guarded fragment with equivalence or transitive relations*, Proc. of Computer Science Logic, 14th Annual Conf. of the EACSL, CSL 2005, LNCS 3634, 309-324.

12 E. Kieroński and M. Otto, *Small substructures and decidability issues for first-order logic with two variables*, Proc. of 20th IEEE Symp. on Logic in Computer Science, LISC 2005, 448-457.

13 E. Kieroński, L. Tendera, *On finite satisfiability of the guarded fragment with equivalence or transitive guards*, Proc. of 14th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2007, LNCS 4790, 318-322.

14 E. Kieroński and L. Tendera, *On finite satisfiability of two-variable first order logic with equivalence relations.*, Proc. of 23rd IEEE Symp. on Logic in Computer Science, LICS 2009, 123-132.

15 H. R. Lewis, *Complexity results for classes of quantificational formulas.* J. Comp. and System Sci., 21:317–353, 1980.

16 C. Lutz, U. Sattler, F. Wolter, *Modal Logic and the two-variable fragment*, Proc. of Computer Science Logic, 10th Annual Conf. of the EACSL, CSL 2001, LNCS 2142, 247-261.

17 A. Manuel, *Two variables and two successors*, Proc. of 35th Int. Symp. on Mathematical Foundations of Computer Science, MFCS 2010, LNCS 6281, 513-524.

18 M. Mortimer, *On languages with two variables*, Zeitschr. f. Logik und Grundlagen d. Mathematik, 21, 1975, 135-140.

19 M. Otto, *Two variable first-order logic over ordered domains*, J. of Symb. Logic, 66, 2001, 685-702.

20 M. Niewerth and T. Schwentick, *Two-variable logic and key constraints on data words*, Proc. of 14th Int. Conf. on Database Theory, ICDT 2011, 138-149.

21 T. Schwentick and T. Zeume, *Two-variable logic with two order relations*, Proc. of Computer Science Logic, 19th Annual Conf. of the EACSL, CSL 2010, LNCS 6247, 499-513.

22 W. Szwast, L. Tendera, *The guarded fragment with transitive guards*, Ann. Pure Appl. Logic 128(1-3), 2004, 227-276.

Coalgebraic Derivations in Logic Programming*

Ekaterina Komendantskaya[1] and John Power[2]

1 **Department of Computing,**
University of Dundee, UK
katya@computing.dundee.ac.uk
2 **Department of Computer Science,**
University of Bath, UK
A.J.Power@bath.ac.uk

Abstract

Coalgebra may be used to provide semantics for SLD-derivations, both finite and infinite. We first give such semantics to classical SLD-derivations, proving results such as adequacy, soundness and completeness. Then, based upon coalgebraic semantics, we propose a new sound and complete algorithm for parallel derivations. We analyse this new algorithm in terms of the Theory of Observables, and we prove soundness, completeness, correctness and full abstraction results.

1998 ACM Subject Classification D.1.6 Logic Programming; F.3.2 Semantics of Programming Languages; F.1.2 Models of Computation

Keywords and phrases Logic programming, SLD-resolution, coalgebra, Lawvere theories, coinductive logic programming, concurrent logic programming

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.352

1 Introduction

In the standard formulations of logic programming, such as in Lloyd's book [19], a first-order logic program P consists of a finite set of clauses of the form $A \leftarrow A_1, \ldots, A_n$, where A and the A_i's are atomic formulae, typically containing free variables, and where $A_1, \ldots, A_n$ is understood to mean the conjunction of the A_i's: note that n may be 0.

A running example of a logic program in this paper is as follows.

▶ **Example 1.1.** Let ListNat denote the logic program

```
        nat(0)  ←
      nat(s(x))  ←  nat(x)
      list(nil)  ←
list(cons x y)  ←  nat(x), list(y)
```

The program involves variables `x` and `y`, function symbols `0`, `s`, `nil` and `cons`, and predicate symbols `nat` and `list`, with the choice of notation designed to make the intended meaning of the program clear.

SLD-resolution, which is a central algorithm for logic programming, takes a goal G, typically written as $\leftarrow B_1, \ldots, B_n$, where the list of B_i's is again understood to mean a

* We acknowledge EPSRC PDRF EP/F044046/2; and the SICSA distinguished visiting fellowship.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 352–366

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

conjunction of atomic formulae, typically containing free variables, and constructs a proof for an instantiation of G from substitution instances of the clauses in P [19]. The algorithm uses Horn-clause logic, with variable substitution determined universally to make a selected atom in G agree with the head of a clause in P, then proceeding inductively. Section 2 recalls the various definitions.

SLD-resolution is sound and complete with respect to least fixed point semantics [19]. But the analysis afforded by least fixed point operators pertains only to finite SLD derivations, whereas infinite SLD derivations are also common in the practice of programming. An example is as follows.

▶ **Example 1.2.** The following program `Stream` defines the infinite stream of binary bits:

```
                bit(0)   ←
                bit(1)   ←
stream(scons (x,y))   ←   bit(x),stream(y)
```

Programs like `Stream` can be given declarative semantics via the *greatest* fixed point of the semantic operator T_P, see also Section 2. But greatest fixed point semantics is incomplete in general [19] as it fails for some infinite derivations.

▶ **Example 1.3.** The program $R(x) \leftarrow R(f(x))$ is characterised by the greatest fixed point of the T_P operator, which contains $R(f^\omega(a))$, but no infinite term is computed by SLD-resolution.

There have been numerous attempts to resolve the mismatch between infinite derivations and greatest fixed point semantics, e.g., [2, 11, 13, 19, 20, 22, 25]. But infinite SLD derivations of both finite and infinite objects have not yet received a uniform semantics, see Figure 1.

In [15, 17], we described an algebraic (fibrational) semantics for logic programming and proved soundness and completeness results for it with respect to SLD-resolution. Other forms of algebraic semantics for logic programming have been given in [1, 5]. Here, we give coalgebraic semantics for both finite and infinite SLD derivations, and prove soundness and completeness results for it, see Sections 3, 4. That constitutes the first main contribution of the paper.

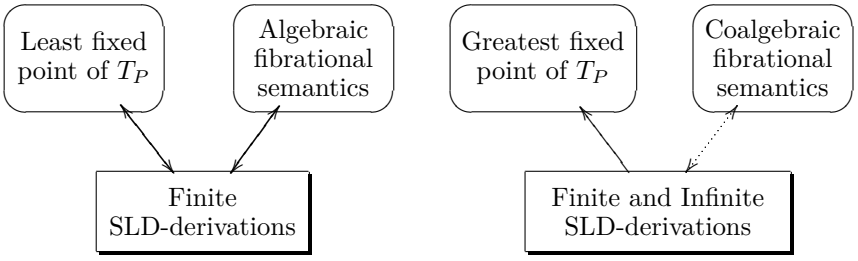


Figure 1 Alternative declarative semantics for finite and infinite SLD-derivations. The arrows $\leftrightarrow$ show the semantics that are both sound and complete, and the arrow $\rightarrow$ indicates sound incomplete semantics. The dotted arrow indicates the sound and complete semantics we propose here.

Another distinguishing feature of logic programming languages is that they allow implicit parallel execution of programs. The three main types of parallelism used in implementations are *and-parallelism*, *or-parallelism*, and their combination: see [12, 23] for analysis.

Or-parallelism arises when more than one clause unifies with the goal: the corresponding bodies can be executed in or-parallel fashion. Or-parallelism is thus a way of efficiently searching for solutions to a goal, by exploring alternative solutions in parallel. It has been exploited in Aurora and Muse, both of which have shown good speed-up results over a considerable range of applications.

And-parallelism arises when more than one atom is present in the goal. That is, given a goal $G = \leftarrow B_1, \ldots B_n$, an *and-parallel algorithm* for SLD resolution looks for SLD derivations for each B_i simultaneously, subject to the condition that the atoms must not share variables. Such cases are known as *independent* and-parallelism. Independent and-parallelism has been successfully exploited in &-PROLOG.

The coalgebraic models we discuss in this paper exhibit a synthetic form of parallelism: *and-or parallelism*. The most common way to express and-or parallelism in logic programs is via *and-or trees* [12], which consist of both *or-nodes* and *and-nodes*. And-or parallel PROLOG works best for variable-free logic programs or DATALOG, and was first implemented in Andorra [7], see also [12]. But many first-order algorithms are P-complete and hence inherently sequential [8, 14]. This especially concerns first-order unification and variable substitution in the presence of variable dependencies. So extensions of and-or parallel derivations to the general case require complicated algorithms that coordinate variable substitution in different branches of and-or parallel derivation trees [12]. If such synchronisation is omitted, parallel SLD-derivations may lead to unsound results, see also Section 5.

In Section 5, we propose an alternative derivation algorithm inspired by our coalgebraic semantics [18]. It inherently models substitutions in a uniform way, so that additional techniques for synchronisation of substitutions are not required. We support the algorithm with soundness, completeness, correctness and full abstraction results with respect to the coalgebraic semantics. That is the second major contribution of the paper.

The underlying category theory of this paper was developed in [18], but the relationship with ordinary logic programming syntax was not systematically developed there, in particular with none of the syntax/semantics results given there.

2 First-order logic programming

We recall some basic definitions from [19].

A *signature* Σ consists of a set of *function symbols* $f, g, \ldots$ each equipped with a fixed *arity*. The arity of a function symbol is a natural number indicating the number of its arguments. Nullary (0-ary) function symbols are allowed: these are called *constants*. Given a countably infinite set Var of variables, the set $Ter(\Sigma)$ of *terms* over Σ is defined inductively: $x \in Ter(\Sigma)$ for every $x \in Var$. If f is an n-ary function symbol ($n \geq 0$) and $t_1, \ldots, t_n \in Ter(\Sigma)$, then $f(t_1, \ldots, t_n) \in Ter(\Sigma)$. Variables will be denoted x, y, z, sometimes with indices $x_1, x_2, x_3, \ldots$. A *substitution* is a map $\theta : Ter(\Sigma) \to Ter(\Sigma)$ which satisfies $\theta(f(t_1, \ldots, t_n)) \equiv f(\theta(t_1), \ldots, \theta(t_n))$ for every n-ary function symbol f.

We define an *alphabet* to consist of a signature Σ, the set Var, and a set of *predicate symbols* $P, P_1, P_2, \ldots$, each assigned an arity. Let P be a predicate symbol of arity n and $t_1, \ldots, t_n$ be terms. Then $P(t_1, \ldots, t_n)$ is a *formula* (also called an atomic formula or an *atom*). The *first-order language* $\mathcal{L}$ given by an alphabet consists of the set of all formulae constructed from the symbols of the alphabet.

Given a substitution θ and an atom A, we write $A\theta$ for the atom given by applying the substitution θ to the variables appearing in A. Moreover, given a substitution θ and a list of atoms $(A_1, ..., A_k)$, we write $(A_1, ..., A_k)\theta$ for the simultaneous substitution of θ in each A_m.

Given a first-order language $\mathcal{L}$, a *logic program* consists of a finite set of clauses of the form $A \leftarrow A_1, \ldots, A_n$, where $A, A_1, \ldots, A_n (\ n \geq 0)$ are atoms. The atom A is called the *head* of a clause, and $A_1, \ldots, A_n$ is called its *body*. Clauses with empty bodies are called *unit clauses*. A *goal* is given by $\leftarrow B_1, \ldots B_n$, where $B_1, \ldots B_n (\ n \geq 0)$ are atoms.

Traditionally, logic programming has been modelled by *least fixed point* semantics [19]. Given a logic program P, one lets B_P (also called a *Herbrand base*) denote the set of atomic ground formulae generated by the syntax of P, and one defines $T_P(I)$ on 2^{B_P} by sending I to the set $\{A \in B_P : A \leftarrow A_1, ..., A_n$ is a ground instance of a clause in P with $\{A_1, ..., A_n\} \subseteq I\}$. The least fixed point of T_P is called the *least Herbrand model* of P and duly satisfies model-theoretic properties that justify that expression [19]. A non-ground alternative to this semantics was further developed in terms of categorical logic in [1, 5].

The fact that logic programs can be represented naturally by least fixed point semantics led to the development of *logic programs as inductive definitions* [22, 13]. Operational semantics for logic programs is given by SLD-resolution, a goal-oriented proof-search procedure.

Let S be a finite set of atoms. A substitution θ is called a *unifier* for S if, for any pair of atoms A_1 and A_2 in S, applying the substitution θ yields $A_1\theta = A_2\theta$. A unifier θ for S is called a *most general unifier* (mgu) for S if, for each unifier σ of S, there exists a substitution γ such that $\sigma = \theta\gamma$.

▶ **Definition 2.1.** Let a goal G be $\leftarrow A_1, \ldots, A_m, \ldots, A_k$ and a clause C be $A \leftarrow B_1, \ldots, B_q$. Then G' is *derived* from G and C using mgu θ if the following conditions hold:

- θ is an *mgu* of the *selected* atom A_m in G and A;
- G' is the goal $\leftarrow (A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k)\theta$.

A clause C_i^* is a *variant* of the clause C_i if $C_i^* = C_i\theta$, with θ being a variable renaming substitution such that variables in C_i^* do not appear in the derivation up to G_{i-1} (see the notation below). This process of renaming variables is called *standardising the variables apart*; we assume it throughout the paper without explicit mention.

▶ **Definition 2.2.** An *SLD-derivation* of $P \cup \{G\}$ consists of a sequence of goals $G = G_0, G_1, \ldots$ called *resolvents*, a sequence $C_1, C_2, \ldots$ of variants of program clauses of P, and a sequence $\theta_1, \theta_2, \ldots$ of mgus such that each G_{i+1} is derived from G_i and C_{i+1} using θ_{i+1}. An *SLD-refutation* of $P \cup \{G\}$ is a finite SLD-derivation of $P \cup \{G\}$ that has the empty clause $\Box$ as its last goal. If $G_n = \Box$, we say that the refutation has length n. The composition $\theta_1, \theta_2, \ldots$ is called *computed answer*.

SLD-resolution is P-complete, and hence inherently sequential [8]. Operationally, SLD-derivations can be characterised by *SLD-trees*.

▶ **Definition 2.3.** Let P be a logic program and G be a goal. An *SLD-tree* for $P \cup \{G\}$ is a tree T satisfying the following:

1. each node of the tree is a (possibly empty) goal
2. the root node is G
3. if $\leftarrow A_1, \ldots, A_m$, $m > 0$ is a node in T, and it has n children, then there exists $A_k \in A_1, \ldots, A_m$ such that A_k is unifiable with exactly n distinct clauses $C_1 = A^1 \leftarrow B_1^1, \ldots, B_q^1$, ..., $C_n = A^n \leftarrow B_1^n, \ldots, B_r^n$ in P via mgus $\theta_1, \ldots \theta_n$, and, for every $i \in \{1, \ldots n\}$, the ith child node is given by the goal

$$\leftarrow (A_1, \ldots, A_{k-1}, B_1^i, \ldots, B_q^i, A_{k+1}, \ldots, A_m)\theta_i$$

4. nodes which are the empty clause have no children.

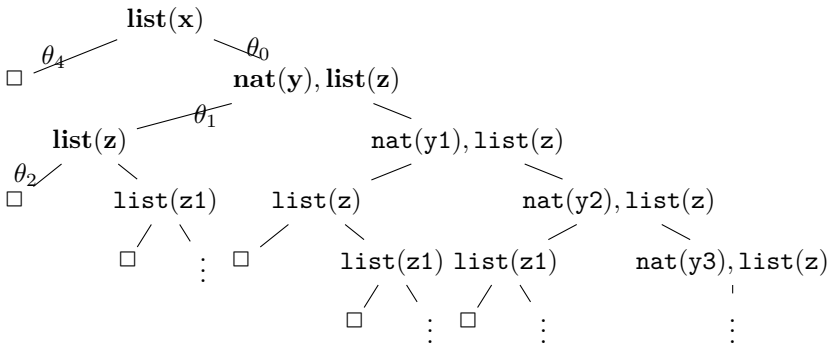


Figure 2 An SLD-tree for ListNat with the goal $\leftarrow list(x)$. A possible computed answer is given by the composition of $\theta_0 = x/cons(y, z)$, $\theta_1 = y/0$, $\theta_2 = z/nil$; Another computed answer is $\theta_4 = x/nil$.

▶ **Example 2.4.** Figure 2 shows an SLD-tree for ListNat (Example 1.1). Note that a similar goal `stream(x)` in the logic program `Stream` from Example 1.2 will produce a very different SLD-tree in that it will not have leaf nodes. The nodes will infinitely alternate between `stream(x)` and `bit(y),stream(z)`, modulo variable renaming.

SLD-resolution is sound and complete with respect to least fixed point semantics. The classical theorems of soundness and completeness of this operational semantics [19] show that every atom in the set computed by the least fixed point of T_P has a finite SLD-refutation, and vice versa.

3 Coalgebraic Semantics for SLD-derivations

Logic programs resemble, and indeed induce, transition systems or rewrite systems, hence coalgebras. That fact has been used to study their operational semantics, e.g., in [4, 6]. In [16], we developed the idea for variable-free logic programs, extending it to first-order programs in [18]. In this section, we recall the relevant details.

Given a set At of atoms, there is a bijection between the set of variable-free logic programs over At and the set of $P_f P_f$-coalgebra structures on At, i.e., functions $p : At \longrightarrow P_f P_f(At)$, where P_f is the finite powerset functor: each atom of a logic program P is the head of finitely many clauses, and the body of each of those clauses contains finitely many atoms.

The endofunctor $P_f P_f$ necessarily has a cofree comonad $C(P_f P_f)$ on it as follows.

▶ Proposition 3.1. Let $C(P_f P_f)$ denote the cofree comonad on $P_f P_f$. For any set At, $C(P_f P_f)(\mathrm{At})$ is the limit of a diagram of the form

$$\ldots \longrightarrow \mathrm{At} \times P_f P_f(\mathrm{At} \times P_f P_f(\mathrm{At})) \longrightarrow \mathrm{At} \times P_f P_f(\mathrm{At}) \longrightarrow \mathrm{At}.$$

Given $p : \mathrm{At} \longrightarrow P_f P_f(\mathrm{At})$, put $\mathrm{At}_0 = \mathrm{At}$ and $\mathrm{At}_{n+1} = \mathrm{At} \times P_f P_f(\mathrm{At}_n)$, and consider the cone defined inductively as follows:

$$\begin{aligned} p_0 &= id : \mathrm{At} \longrightarrow \mathrm{At}\ (= \mathrm{At}_0) \\ p_{n+1} &= \langle id, P_f P_f(p_n) \circ p \rangle : \mathrm{At} \longrightarrow \mathrm{At} \times P_f P_f(\mathrm{At}_n)\ (= \mathrm{At}_{n+1}) \end{aligned}$$

The limiting property determines the coalgebra $\overline{p} : \mathrm{At} \longrightarrow C(P_f P_f)(\mathrm{At})$.

The main result of [16] asserted that if $C(P_f P_f)$ is the cofree comonad on $P_f P_f$, then, given a logic program P, the induced $C(P_f P_f)$-coalgebra structure characterises the parallel and-or derivation trees (cf. [12]) of P.

▶ **Example 3.2.** Consider the variable-free logic program: `q(b,a)` ←; `s(a,b)` ←; `p(a)` ← `q(b,a)`, `s(a,b)`; `q(b,a)` ← `s(a,b)`.

The program has three atoms, namely `q(b,a)`, `s(a,b)` and `p(a)`.
So $At = \{\texttt{q(b,a)}, \texttt{s(a,b)}, \texttt{p(a)}\}$. The program can be identified with the $P_f P_f$-coalgebra structure on At given by
$p(\texttt{q(b,a)}) = \{\{\}, \{\texttt{s(a,b)}\}\}$, where $\{\}$ is the empty set.
$p(\texttt{s(a,b)}) = \{\{\}\}$, i.e., the one element set consisting of the empty set.
$p(\texttt{p(a)}) = \{\{\texttt{q(b,a)}, \texttt{s(a,b)}\}\}$.

Consider the $C(P_f P_f)$-coalgebra corresponding to p. It sends `p(a)` to the parallel refutation of `p(a)` depicted on the left side of Figure 3. Note that the nodes of the tree alternate between those labeled by atoms and those labeled by bullets (•). The set of children of each bullet represents a goal, made up of the conjunction of the atoms in the labels. An atom with multiple children is the head of multiple clauses in the program: its children represent these clauses. We use the traditional notation □ to denote $\{\}$.

Where an atom has a single •-child, we can elide that node without losing any information; the result of applying this transformation to our example is shown on the right in Figure 3. The resulting tree is precisely the parallel and-or derivation tree [12] for the atomic goal ← p(a).

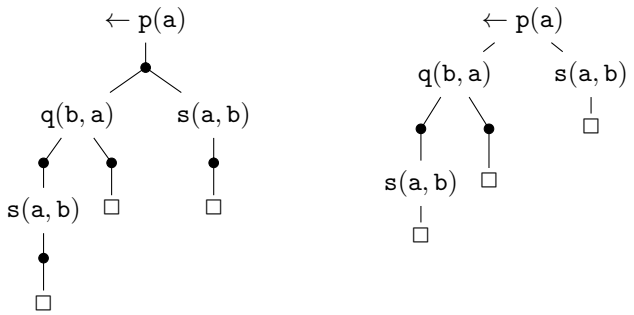


Figure 3 The action of $\overline{p} : \mathrm{At} \longrightarrow C(P_f P_f)(\mathrm{At})$ on `p(a)`, and the corresponding parallel and-or derivation tree [12].

In [18], we extend this to first-order logic programs using *Lawvere theories*, cf, [1, 4, 5, 15], modelling most general unifiers (mgu's) by *equalisers*. cf, [3]: given a signature Σ, the Lawvere theory $\mathcal{L}_\Sigma$ generated by Σ has objects given by natural numbers and maps from n to m given by equivalence classes of substitutions θ of m variables by terms generated by the function symbols in Σ applied to n variables. We shall shortly give an example; for formal definitions and theorem see [18].

Given a logic program P with function symbols in Σ, we extend the set At of atoms in a variable-free logic program to the functor from $\mathcal{L}_\Sigma^{op}$ to Set sending a natural number n to the set $At(n)$ of atomic formulae with at most n variables generated by the predicate symbols in P. One can extend any endofunctor H on Set to the endofunctor $[\mathcal{L}_\Sigma^{op}, H]$ on $[\mathcal{L}_\Sigma^{op}, Set]$ that sends $F : \mathcal{L}_\Sigma^{op} \to Set$ to the composite HF. We would like to model P by the putative $[\mathcal{L}_\Sigma^{op}, P_f P_f]$-coalgebra $p : At \longrightarrow P_f P_f At$ that, at n, takes an atomic formula $A(x_1, \ldots, x_n)$ with at most n variables, considers all substitutions of clauses in P whose head agrees with $A(x_1, \ldots, x_n)$, and gives the set of sets of atomic formulae in antecedents, mimicking the construction for variable-free logic programs.

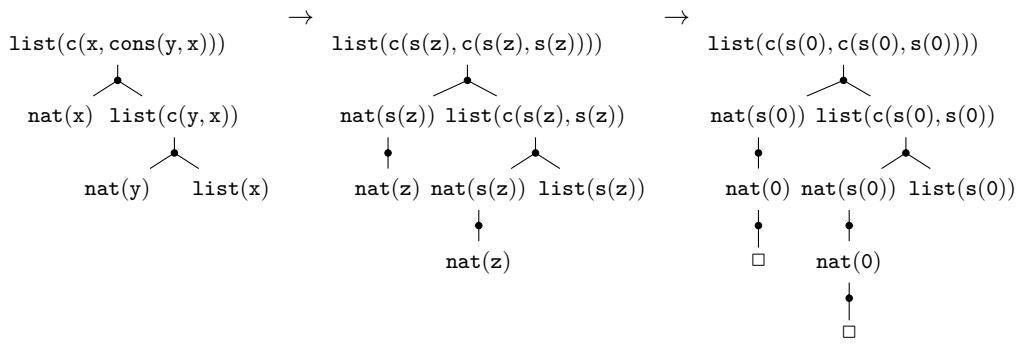


Figure 4 The left hand tree represents $\bar{p}(\texttt{list}(\texttt{cons}(\texttt{x}, \texttt{cons}(\texttt{y}, \texttt{x}))))$ and the second tree represents $\bar{p}At((s,s))(\texttt{list}(\texttt{cons}(\texttt{x}, \texttt{cons}(\texttt{y}, \texttt{x}))))$, i.e., $\bar{p}(\texttt{list}(\texttt{cons}(\texttt{s}(\texttt{z}), \texttt{cons}(\texttt{s}(\texttt{z}), \texttt{s}(\texttt{z})))))$, and the tree on the right depicts $\bar{p}At(0)At((s,s))(\texttt{list}(\texttt{cons}(\texttt{x}, \texttt{cons}(\texttt{y}, \texttt{x}))))$; cons is abbreviated by c.

In fact, to make the theory work, we need to extend Set to $Poset$, natural transformations to *lax natural transformations*, and replace the outer instance of P_f by P_c - the countable powerset functor (as recursion generates countability). Subject to those replacements, $p : At \longrightarrow P_c P_f At$ behaves as above, giving a $Lax(\mathcal{L}_\Sigma^{op}, P_c P_f)$-coalgebra structure on At. Extending Proposition 3.1, p determines a $Lax(\mathcal{L}_\Sigma^{op}, C(P_c P_f))$-coalgebra structure $\bar{p} : At \longrightarrow C(P_c P_f)(At)$.

▶ **Example 3.3.** Consider ListNat as in Example 1.1. Suppose we start with $A(x,y) \in At(2)$ given by the atomic formula $\texttt{list}(\texttt{cons}(\texttt{x}, \texttt{cons}(\texttt{y}, \texttt{x})))$. Then $\bar{p}(A(x,y))$ is the element of $C(P_c P_f)At(2)$ expressible by the tree on the left hand side of Figure 4.

The coalgebraic structure means any substitution, whether determined by an mgu or not, applies to the whole tree. The lax naturality means a substitution potentially yields two different trees: one given by substitution into the tree, then pruning to remove redundant branches, the other given by substitution into the root, then applying $\bar{p}$.

For example, we can substitute $s(z)$ for both x and y in $\texttt{list}(\texttt{cons}(\texttt{x}, \texttt{cons}(\texttt{y}, \texttt{x})))$. This substitution is given by applying At to the map $(s,s) : 1 \longrightarrow 2$ in $\mathcal{L}_\Sigma$. So $At((s,s))(A(x,y))$ is an element of $At(1)$. Its image under $\bar{p}(1) : At(1) \longrightarrow C(P_c P_f)At(1)$ is the element of $C(P_c P_f)At(1)$ expressible by the tree on the right hand side of Figure 4. The laxness of the naturality of $\bar{p}$ is indicated by the increased length, in two places, of the second tree. Observe that, before those two places, the two trees have the same structure: that need not always be exactly the case, as substitution in a tree could involve pruning if substitution instances of two different atoms yield the same atom.

Now suppose we make the further substitution of 0 for z. This substitution is given by applying At to the map $0 : 0 \to 1$ in $\mathcal{L}_\Sigma$. In Figure 4, we depict $\bar{p}(0)At(0)At((s,s))(A(x,y))$ on the right. Two of the leaves of the latter tree are labeled by $\Box$, but one leaf, namely $\texttt{list}(\texttt{s}(0))$ is not, so the tree does not yield a proof. Again, observe the laxness.

The trees shown in Example 3.3 differ from the corresponding SLD-tree determined by Definition 2.3. The main reason for this is that the derivations modelled by the coalgebraic semantics have strong relation to *parallel logic programming*, [26, 14], while SLD-trees describe sequential derivation strategies.

In following sections, we shall show how our coalgebraic semantics relates to sequential derivations, and how it can be used to introduce a new concurrent derivation algorithm.

4 Coalgebraic Semantics and Infinite Derivations

In this section, we state formally the theorems that relate the coalgebraic semantics of the previous section to first-order (possibly infinite) derivations in logic programming. We start by introducing a special kind of derivation tree that is suitable for representing derivations described by the coalgebraic semantics.

▶ **Definition 4.1.** Let P be a logic program and $G =\leftarrow A$ be an atomic goal. The *coinductive derivation tree* for A is a possibly infinite tree T satisfying the following properties.

- A is the root of T.
- Each node in T is either an and-node or an or-node.
- Each or-node is given by $\bullet$.
- Each and-node is an atom.
- For every and-node A' occurring in T, there exist exactly $m > 0$ distinct clauses $C_1, \ldots, C_m$ in P (a clause C_i has the form $B_i \leftarrow B_1^i, \ldots, B_{n_i}^i$, for some n_i), such that $A' = B_1\theta_1 = \ldots = B_m\theta_m$, for some substitutions $\theta_1, \ldots, \theta_m$, then A' has exactly m children given by or-nodes, such that, for every $i \in m$, the ith or-node has n children given by and-nodes $B_1^i\theta_i, \ldots, B_{n_i}^i\theta_i$.

▶ **Example 4.2.** Examples of coinductive derivation trees are given in Figures 3 and 4.

Note that, comparing this with the SLD-resolution algorithm and the corresponding SLD-trees, the definition of coinductive derivation tree restricts unification to the case of *term matching*, i.e., the substitution θ unifying atoms A_1 and A_2 is applied only to one atom, e.g. $A_1 = A_2\theta$, whereas traditionally mgus satisfy $A_1\theta = A_2\theta$. The term-matching algorithm is parallelisable, in contrast to the unification algorithm, which is inherently sequential [8].

We define the *depth of a coinductive tree* inductively as follows. The root of a coinductive tree has depth 0. For an and-node x, if its immediate parent and-node has depth d, then x has depth $d + 1$. The depth of a tree is defined to be the depth of its deepest branch.

For all the running examples we use in this paper, there will be only one coinductive tree for every goal. However, this will not be the case for programs containing clauses in which not all the variables appearing in the body appear in the head.

▶ **Example 4.3.** In [16] we analyse the program determining whether two nodes in a graph are connected. It contains the clause $\texttt{connected}(x, y) \leftarrow \texttt{edge}(x, z), \texttt{connected}(z, y)$, note the appearance of z.

According to Deginition 4.1 such clauses may induce a family of coinductive trees - as there can be a countable number of substitutions $\theta_i', \ldots, \theta_i''$ that match a given goal with the clause C_i, each of these substitutions differing only with respect to assignment to z.

▶ **Definition 4.4.** Let P be a logic program and $G =\leftarrow A$ be an atomic goal. The *coinductive forest* F for A is a set of all coinductive derivation trees for A. We say that the forest has *depth* n if the deepest tree in F has length n. A coinductive forest F has *breadth* k if at most k distinct variables appear in all and-nodes of all of its trees together.

▶ **Theorem 4.5** (Adequacy). *For any logic program P and for any atom A generated by the predicate symbols of P and k distinct variables $x_1, \ldots, x_k$, $\bar{p}(k)(A)$ expresses precisely the same information as that given by a coinductive forest F for the goal A. That is, the following holds:*

- *$p_n(k)(A)$ is isomorphic to the coinductive forest of depth n and breadth k.*
- *F has the finite depth n if and only if $\bar{p}(k)(A) = p_n(k)(A)$.*

- *F has infinite depth if and only if $\bar{p}(k)(A)$ is given by the element of the limit of the infinite chain given by (the extension of) Proposition 3.1.*

Proof. By (the extension of) Proposition 3.1, for every atomic formula A:
- $p_0(k)(A) = A$
- $p_1(k)(A) = (A, \{\{B^1\theta, \ldots, B^m\theta\}$, such that $B \leftarrow B^1, \ldots, B^m$ is a clause in P with $B\theta = A$ and $B^1\theta, \ldots, B^m\theta$ have variables among $x_1, \ldots, x_k.\})$
- $p_2(k)(A) = (A, \{\{(B^1\theta, \{\{C_1^1\theta_1\theta, \ldots, C_1^{m_1}\theta_1\theta\}$ such that $C \leftarrow C_1^1, \ldots, C_1^{m_1}$ is a clause in P with $C\theta_1 = B^1\}), \ldots$ and $C_1^1\theta_1\theta, \ldots, C_1^{m_1}\theta_1\theta$ have variables among $x_1, \ldots, x_k.\}\})$
- etc.

The limit of the sequence is precisely (the extension of) the structure described by Proposition 3.1. For each atomic formula A, $p_0(k)(A)$ corresponds to the root of a coinductive derivation tree, and, more generally, each $p_n(k)(A)$ corresponds to the coinductive forest of breadth k, as far as depth n. ◀

▶ **Example 4.6.** Infinite coinductive trees arise in programs similar to that in Example 1.3. The infinite tree arising from this program contains a chain of alternating $\bullet$'s and atoms $R(x)$, $R(f(x))$, $R(f(f(x)))$, etcetera. Note that infinite terms are not nodes of the tree. Programs like `Stream` and `ListNat` in Examples 1.1 and 1.2, do not give rise to infinite coinductive derivation trees, see Figures 4 and 6. But they do give rise to infinite SLD-trees, see Figure 2. This is because substitution, determined by term-matching, is applied only to clauses, and not to goals, when a coinductive derivation tree is built. Infinite derivations in these programs may be modelled by infinite chains of derivation trees.

We can express Theorem 4.5 in terms of a traditional-style soundness and completeness result that relates the semantics to SLD-refutations. For this purpose, we define *success subtrees* of coinductive derivation trees, as follows.

▶ **Definition 4.7.** Let P be a logic program, A be a goal, and T be the coinductive derivation tree determined by P and A. A subtree T' of T is called a *success subtree* of T if it satisfies the following conditions:
- the root of T' is the root of T;
- if an and-node belongs to T', and the node has k children in T given by or-nodes, only one of these or-nodes belongs to T'.
- if an or-node belongs to T', then all its children given by and-nodes in T belong to T'.
- all the leaves of T' are and-nodes represented by $\Box$.

▶ **Theorem 4.8** (Soundness and Completeness of SLD-resolution relative to coinductive derivation trees.)**.** *Let P be a logic program, and G be a goal.*
1. *Soundness. If there is an SLD-refutation for G in P with computed answer θ, then there exists a coinductive derivation tree for $G\theta$ that contains a success subtree.*
2. *Completeness. If a coinductive derivation tree for $G\theta$ contains a success subtree, then there exists an SLD-refutation for G in P, with computed answer λ such that there exists substitution σ such that $\lambda = \sigma\theta$.*

Proof. The proof is given by induction on the length of the SLD-refutations and the depth of the coinductive trees. Part 2 also requires some analysis of computed answers. If a program does not contain clauses similar to Example 4.3, then σ is an identity substitution or a variable renaming, otherwise σ is determined by all the substitutions computed by the SLD-derivations that involved assigning terms to the variables appearing in the body but not the head of clauses in P. ◀

▶ **Corollary 4.9** (Soundness and Completeness of SLD-resolution relative to coalgebraic semantics)**.** *Given a logic program P, SLD-refutations in P are sound and complete with respect to the $Lax(\mathcal{L}_\Sigma^{op}, P_c P_f)$-coalgebra determined by P.*

Proof. Follows from Theorems 4.5 and 4.8. ◀

Our coalgebraic analysis relates to the *Theory of Observables* for logic programming developed in [6]. In that theory, the traditional characterisation of logic programs in terms of input/output behavior and successful derivations is not sufficient for the purposes of program analysis and optimisation. One requires more complete information about SLD-derivations, e.g., the sequences of goals, most general unifiers, and variants of clauses. Moreover, infinite derivations can be meaningful. The following four observables are the most important for the theory [9, 6].

▶ **Definition 4.10.** 1. *Partial answers* are the substitutions associated to a resolvent in any SLD-derivation; *correct partial answers* are substitutions associated to a resolvent in any SLD-refutation.
2. *Call patterns* are atoms selected in any SLD-derivation; *correct call patterns* are atoms selected in any SLD-refutation.
3. *Computed answers* are the substitutions associated to an SLD-refutation.
4. *A successful derivation* is the observation of successful termination.

As argued in [9, 6], a key goal of semantics to logic programs is to observe equal behavior of logic programs and to distinguish logic programs with different computational behavior. The choice of observables and semantic models is closely related to the choice of equivalence relation defined over logic programs [9].

▶ **Definition 4.11.** Let P_1 and P_2 be logic programs. Put $P_1 \approx P_2$ if and only if, for a goal G, the following four conditions hold:
1. G has a refutation in P_1 if and only if G has a refutation in P_2
2. G has the same set of computed answers in P_1 as in P_2
3. G has the same set of (correct) partial answers in P_1 as in P_2
4. G has the same set of call patterns in P_1 as in P_2.

Using the terminology of [9, 6], we can state the following *correctness result* that relates the traditional sequential SLD-derivations of Section 2 to our coalgebraic semantics. In the next theorem, we assume that there is a common algorithm that assigns terms to variables appearing only in the bodies of clauses as explained in Example 4.3.

▶ **Theorem 4.12** (Correctness)**.** *For logic programs P_1 and P_2, if for every atomic goal $\leftarrow A$, the coinductive forest for P_1 and A is equal to the coinductive forest for P_2 and A, then $P_1 \approx P_2$.*

The converse of Theorem 4.12, the *full abstraction result*, does not hold. That is, there can be observationally equivalent programs that have different coinductive derivation trees.

▶ **Example 4.13.** Consider the logic programs P_1 and P_2, whose clauses are the same, with the exception of one clause: P_1 contains $A \leftarrow B_1, \ldots, B_i, \texttt{false}, \ldots, B_n$; and P_2 contains the clause $A \leftarrow B_1, \ldots B_i, \texttt{false}$ instead. The atoms in the clauses are such that $B_1, \ldots, B_i$ have refutations in P_1 and P_2, and `false` is an atom that has no refutation in the programs. In this case, assuming a left-to-right sequential evaluation strategy, all derivations that involve the two clauses in P_1 and P_2 will always fail on `false`, and P_1 will be observationally equivalent to P_2. However, their coinductive derivation trees give account to all atoms in the clause.

The results of this section show that parallel trees arising from the coalgebraic semantics of Section 3 naturally model finite and infinite derivations. The nature of the failure of the full abstraction result suggests that the coalgebraic semantics of Section 3 more naturally supports concurrent computation, rather than sequential SLD-derivations. For this reason, we introduce a novel algorithm for concurrent derivations in the next section.

5 Applications in Concurrent Logic Programming

In this section, we exploit the concurrent nature of our coalgebraic semantics, equivalently coinductive derivation trees. Operationally, the major difference between coinductive trees and SLD-trees lies in the concurrent versus sequential modes of execution, which are crucial for the computation of call patterns, (correct) partial answers and soundness of computations.

We first consider a concurrent computational model already in the literature: and-or-parallel trees [12].

▶ **Definition 5.1.** [12] Let P be a logic program and let $\leftarrow A$ be an atomic goal (possibly with variables). The *and-or parallel derivation tree* for A is the possibly infinite tree T satisfying the following properties.

- A is the root of T.
- Each node in T is either an and-node or an or-node.
- Each or-node is given by $\bullet$.
- Each and-node is an atom.
- For every node A' occurring in T, if A' is unifiable with only one clause $B \leftarrow B_1, \ldots, B_n$ in P with mgu θ, then A' has n children given by and-nodes $B_1\theta, \ldots B_n\theta$.
- For every node A' occurring in T, if A' is unifiable with exactly $m > 1$ distinct clauses $C_1, \ldots, C_m$ in P via mgu's $\theta_1, \ldots, \theta_m$, then A' has exactly m children given by or-nodes, such that, for every $i \in m$, if $C_i = B^i \leftarrow B^i_1, \ldots, B^i_n$, then the ith or-node has n children given by and-nodes $B^i_1\theta_i, \ldots, B^i_n\theta_i$.

An example of an and-or tree is given in Figure 3. Example 3.2 demonstrates and [16] formally proves that coinductive trees and and-or trees produce the same results in the variable-free case. However, a naive extension of Definition 5.1 to the first-order case yields inconsistent derivations.

▶ **Example 5.2.** Figure 5 shows the and-or parallel tree that finds a refutation $\theta = \{x/0, y/0, x/nil\}$ for the goal `list(cons(x,cons(y,x)))`, although this answer is not sound.

A solution proposed in [12] was given by *composition (and-or parallel) trees*. Construction of composition trees involves additional algorithms that synchronise substitutions in the branches of and-or trees. Composition trees contain a special kind of *composition nodes* used whenever both and- and or-parallel computations are possible for one goal. A composition node is a list of atoms in the goal. If, in a goal $G = \leftarrow B_1, \ldots B_n$, an atom B_i is unifiable with $k > 1$ clauses, then the algorithm adds k children (k composition nodes) to the node G; similarly for every atom in G that is unifiable with more than one clause. Every such composition node has the form $B_1, \ldots B_n$, and n and-parallel edges. Thus, all possible combinations of all possible or-choices at every and-parallel step are given.

Here, we propose coinductive trees of Definition 4.1 as an alternative to composition trees. Comparing coinductive derivation trees with and-or trees, coinductive trees are more intrinsic: and-or parallel trees have most general unifiers built into a single tree, whereas,

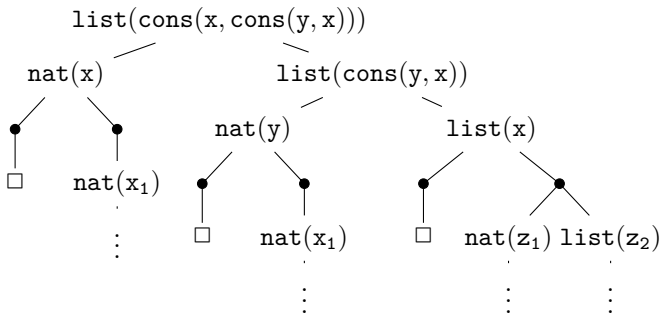


Figure 5 Unsound refutation by and-or parallel tree, with $\theta = \{x/0, y/0, x/nil\}$.

mgus determine only tree transformations for coinductive trees. Taking unification issues from the level of individual leaves to the level of trees affects computations at least in two ways. Parallel proof-search in branches of a coinductive tree does not require synchronisation of variables in different branches. Moreover, for programs that are guarded by constructors - such as `ListNat` and `Stream`, we avoid having infinite branches or infinite number of variables in a single tree. We shall illustrate with our leading example.

▶ **Example 5.3.** The coinductive trees from Figure 4 agree with the first part of the and-or parallel tree for `list(cons(x, cons(y, x)))` in Figure 5. But the coinduction tree has leaves `nat(x)`, `nat(y)` and `list(x)`, whereas the and-or tree follows those nodes, using substitutions determined by mgu's. Moreover, those substitutions need not be consistent with each other: not only are there two ways to unify each of `nat(x)`, `nat(y)` and `list(x)`, but also there is no consistent substitution for `x` at all. In contrast, the coinduction trees capture such cases.

We can go further and introduce a new derivation algorithm that allows proof search using coinduction trees. We modify the definition of a goal by taking it to be a pair $< A, T >$, where A is an atom, and T is the coinduction tree determined by A, as in Definition 4.1, in which we restrict the choice of substitutions $\theta_1, \ldots \theta_m$ to the most general unifiers only, in which case T is uniquely determined by A.

▶ **Definition 5.4.** Let G be a goal given by an atom $\leftarrow A$ and the coinductive tree T induced by A, and let C be a clause $H \leftarrow B_1, \ldots, B_n$. Then goal G' is *coinductively derived* from G and C using mgu θ if the following conditions hold:

- A' is a leaf atom, called the *selected* atom, in T.
- θ is an *mgu* of A' and H.
- G' is given by the atom $\leftarrow A\theta$ and the coinduction tree T' determined by $A\theta$.

▶ **Definition 5.5.** A *coinductive derivation* of $P \cup \{G\}$ consists of a sequence of goals $G = G_0, G_1, \ldots$ called *coinductive resolvents* and a sequence $\theta_1, \theta_2, \ldots$ of mgus such that each G_{i+1} is derived from G_i using θ_{i+1}. A *coinductive refutation* of $P \cup \{G\}$ is a finite coinductive derivation of $P \cup \{G\}$ such that its last goal contains a success subtree. If G_n contains a success subtree, we say that the refutation has length n.

Coinductive derivations resemble *tree rewriting*. In applying SLD-derivation, one's primary interest lies in derivations of atomic goals. But in order to make the induction work, one must generalise goals from being atoms to being lists of atoms, see Definition 2.1. In coinductive tree, this information would be represented by a list of nodes in a truncation of

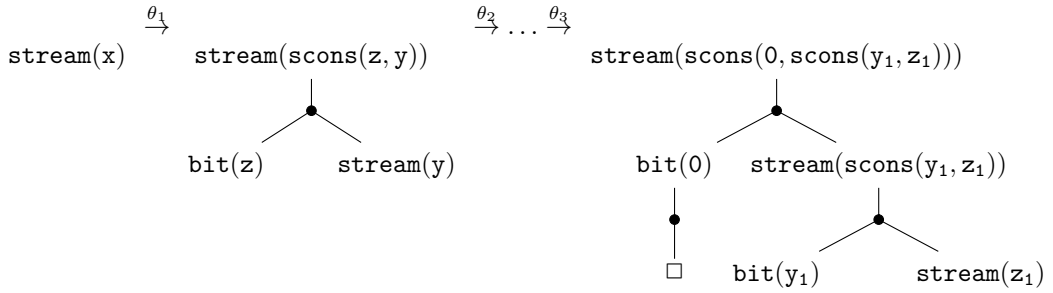


Figure 6 Coinductive derivation of length 3 for the goal $G =$ `stream(x)` and the program `Stream`, with $\theta_1 = x/cons(z, y)$ and $\theta_2 = z/0, \theta_3 = y/cons(y_1, z_1)$.

the coinductive tree. To analyse coinductive derivations, we generalise the definition of a goal a little further, extending it from being an atom A to being the coinductive derivation tree for A, see Definition 4.1. For every goal $G =< A, T >$, there can be several transitions to a new goal, and these transitions can be made concurrently.

▶ **Example 5.6.** Figure 6 shows a coinductive derivation of length 3 for the goal $G =$ `stream(x)` and the program `Stream` from Example 1.2.

▶ **Theorem 5.7** (Soundness and Completeness of coinductive resolution relative to coalgebraic semantics.)**.** *Let P be a program built over the signature Σ, and $G =< A(\bar{t}), T >$ be a goal.*

1. *Soundness. If there is a coinductive derivation of length n of $P \cup \{G\}$ with an answer $\theta = \theta_1 \circ \ldots \circ \theta_n$, and if $G_n =< A_n(\overline{t_n}), T_n >$, $\overline{t_n}$ having k distinct variables, then $\overline{t_n} = \bar{t}\theta$ and $\bar{p}(k)(At(\theta))(A(\bar{t}))$ is isomorphic to the coinductive forest F of breadth k determined by A_n.*
2. *Completeness. Given the $Lax(\mathcal{L}_\Sigma^{op}, P_c P_f)$-coalgebra structure $\bar{p}$ generated by P, let θ be a map in $\mathcal{L}_\Sigma^{op}$, and let C be the structure determined by evaluating $\bar{p} : At(\theta) \longrightarrow C(P_c P_f)(At)$ at a natural number k and applying it to an atomic formula $A(x_1, \ldots, x_k)$. Then there exists a derivation from $G =< A(x_1, \ldots, x_k), T >$ to $G_n =< A_n, T_n >$, with $A_n = A(x_1, \ldots x_k)\sigma$, such that there exists a substitution ρ such that $\theta = \sigma\rho$ and the coinductive forest for $A_n\rho$ is isomorphic to C.*

Proof. The proof proceeds by induction on the length of derivations, using the constructions of Theorem 4.5. ◀

Theorem 5.7 characterises all derivations, not only finite ones, although it can be restricted to coinductive refutations. In general, there are two levels of computation at which both infinity and concurrency can be implemented in coinductive derivations. One level is that of the coinduction trees given by the goals; and the second level is the transitions between the goals. Depending on the applications and resources for parallelisation, the coinductive derivation algorithm above offers several choices as follows.

Every coinductive tree in a goal is necessarily concurrent, but transitions between coinduction trees can be done in a sequential or a concurrent manner. That is, if there are several non-empty leaves in a tree, any such leaf can be unified with some clause in P. Such leaves can provide substitutions for sequential or concurrent tree transitions. In Figure 6, the substitution $\theta' = \theta_2\theta 3$ is derived by considering mgus for two leaves in $G_1 =<$ `stream(scons(z,y))`$, T_1 >$; but, although two separate and-leaves were used to compute θ', θ' was computed by composing the two substitutions sequentially, and only one

tree, T_3, was produced. However, we could concurrently derive two trees from T_2 instead, $G'_2 =< \texttt{stream(scons(0,y))}, T_2 >$ and $G''_2 =< \texttt{stream(scons(z,scons(}y_1\texttt{,}z_1\texttt{)))}, T'_2 >$.

There are choices concerning how to treat infinite coinductive trees arising in derivations. As Example 4.6 shows, some definitions of infinite objects do not give rise to infinite coinduction trees, e.g., `Stream` gives rise to an infinite sequence of finite coinduction trees, cf. Figure 6. This applies equally to any (potentially) infinite data defined using *constructors*, such as `scons` in `Stream` or `cons` and `nil` in `ListNat`. So one may view infinite coinduction trees as "bad" cases, in which (co)recursion is *not guarded by constructors*. In this case, one might decide to halt any derivation of this kind, and amend the program before proceeding. Alternatively, one may decide to prune infinite branches, and continue to look for derivations in other or-branches for the same unchanged logic program.

Finally, as Figure 6 shows, coinductive programs such as `Stream` may give rise to infinite derivations of coinduction trees, in which case implementation may prune the chain of derivations as [11, 25] suggest, or, if infinite production of new streams is desirable, let the coinductive derivations run.

We can now remedy the *full abstraction* result that we have proven to fail for the SLD-derivations, see Section 5. We once again characterise coinductive derivations from the point of view of the Theory of Observables. In particular, we can routinely adapt Definitions 4.10 and 4.11 to coinductive derivations using substitutions and call patterns determined by coinductive derivations rather than by SLD-derivations. Then the following correctness and full abstraction results hold.

▶ **Theorem 5.8.** *$P_1 \approx P_2$ if and only if the $Lax(\mathcal{L}_\Sigma^{op}, P_c P_f)$-coalgebra structure generated by P_1 is equivalent to the $Lax(\mathcal{L}_\Sigma^{op}, P_c P_f)$-coalgebra structure generated by P_2.*

Proof. (Sketch.) Proof proceeds by induction on constructions described in Theorems 4.5 and 5.7. ◀

6 Conclusions and Further Work

The analysis of this paper can be extended to more expressive logic programming languages, such as [10, 24, 21], also to functional programming languages in the style of [22, 2]. We deliberately chose our running examples to correspond to definitions of inductive or coinductive types in such languages.

The key fact driving our analysis has been the observation that the implication $\leftarrow$ acts at a meta-level, like a sequent rather than a logical connective. That observation extends to first-order fragments of linear logic and the Logic of Bunched Implications [10, 24]. So we plan to extend the work in the paper to logic programming languages based on such logics.

The situation regarding higher-order logic programming languages such as λ-*PROLOG* [21] is more subtle. Despite their higher-order nature, such logic programming languages typically make fundamental use of sequents. So it may well be fruitful to consider modelling them in terms of coalgebra too, albeit probably on a sophisticated base category such as a category of Heyting algebras.

References

1 G. Amato, J. Lipton, and R. McGrail. On the algebraic structure of declarative programming languages. *Theor. Comput. Sci.*, 410(46):4626–4671, 2009.

2 D. Ancona, G. Lagorio, and E. Zucca. Type inference by coinductive logic programming. In *TYPES 2008*, volume 5497 of *LNCS*, pages 1–18. Springer, 2009.

3 A. Asperti and S. Martini. Projections instead of variables: A category theoretic interpretation of logic programs. In *ICLP*, pages 337–352, 1989.

4 F. Bonchi and U. Montanari. Reactive systems, (semi-)saturated semantics and coalgebras on presheaves. *Theor. Comput. Sci.*, 410(41):4044–4066, 2009.

5 R. Bruni, U. Montanari, and F. Rossi. An interactive semantics of logic programming. *TPLP*, 1(6):647–690, 2001.

6 M. Comini, G. Levi, and M. C. Meo. A theory of observables for logic programs. *Inf. Comput.*, 169(1):23–80, 2001.

7 V. S. Costa, D. H.D. Warren, and R. Yang. Andorra-I: A parallel prolog system that transparently exploits both and- and or-parallelism. In *PPOPP*, pages 83–93, 1991.

8 C. Dwork, P.C. Kanellakis, and J.C. Mitchell. On the sequential nature of unification. *Journal of Logic Programming*, 1:35–50, 1984.

9 M. Gabrielli, G. Levi, and M.C. Meo. Observable behaviors and equivalnences of logic programs. *Information and Computation*, 122(1):1–29, 1995.

10 J.-Y. Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

11 G. Gupta, A. Bansal, R. Min, L. Simon, and A. Mallya. Coinductive logic programming and its applications. In *ICLP 2007*, volume 4670 of *LNCS*, pages 27–44. Springer, 2007.

12 G. Gupta and V.S. Costa. Optimal implementation of and-or parallel prolog. In *Conference proceedings on PARLE'92*, pages 71–92, New York, NY, USA, 1994. Elsevier North-Holland.

13 M. Jaume. On greatest fixpoint semantics of logic programming. *J. Log. Comput.*, 12(2):321–342, 2002.

14 P. C. Kanellakis. Logic programming and parallel complexity. In *Foundations of Deductive Databases and Logic Programming.*, pages 547–585. M. Kaufmann, 1988.

15 Y. Kinoshita and A. J. Power. A fibrational semantics for logic programs. In *Proceedings of the Fifth International Workshop on Extensions of Logic Programming*, volume 1050 of *LNAI*. Springer, 1996.

16 E. Komendantskaya, G. McCusker, and J. Power. Coalgebraic semantics for parallel derivation strategies in logic programming. In *Proc. of AMAST'2010 - 13th Int. Conf. on Algebraic Methodology and Software Technology*, volume 6486 of *LNCS*, 2010.

17 E. Komendantskaya and J. Power. Fibrational semantics for many-valued logic programs: Grounds for non-groundness. In *JELIA*, volume 5293 of *LNCS*, pages 258–271, 2008.

18 E. Komendantskaya and J. Power. Coalgebraic semantics for derivations in logic programming. In *CALCO'11*, 2011.

19 J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition, 1987.

20 Zoran Majki? Coalgebraic semantics for logic programming. In *18th Workshop on (Constraint) Logic Programming, WLP 2004, March 04-06*, 2004.

21 D. Miller and G. Nadathur. Higher-order logic programming. In *ICLP*, pages 448–462, 1986.

22 L.C. Paulson and A.W. Smith. Logic programming, functional programming, and inductive definitions. In *ELP*, pages 283–309, 1989.

23 E. Pontelli and G. Gupta. On the duality between or-parallelism and and-parallelism in logic programming. In *Euro-Par*, pages 43–54, 1995.

24 D.J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic Series*. Kluwer Academic Publishers, 2002.

25 L. Simon, A. Bansal, A. Mallya, and G. Gupta. Co-logic programming: Extending logic programming with coinduction. In *ICALP*, volume 4596 of *LNCS*, pages 472–483. Springer, 2007.

26 J.D. Ullman and A.V.Gelder. Parallel complexity of logical query programs. *Algorithmica*, 3:5–42, 1988.

Trees in trees: is the incomplete information about a tree consistent? *

Eryk Kopczyński

Institute of Informatics, University of Warsaw
erykk@mimuw.edu.pl

Abstract

We are interested in the following problem: given a tree automaton $\mathcal{A}$ and an incomplete tree description P, does a tree T exist such that T is accepted by $\mathcal{A}$ and consistent with P? A tree description is a tree-like structure which provides incomplete information about the shape of T. We show that this problem can be solved in polynomial time as long as $\mathcal{A}$ and the set of possible arrangements that can be forced by P are fixed. We show how our result is related to an open problem in the theory of incomplete XML information.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases XML, tree automata, incomplete tree descriptions, Euler cycle

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.367

1 Introduction

In [2] and [9], the authors study the problem of incomplete data in relational databases, and classify the complexity of various computational problems associated with incompleteness, like consistency. These results have become very influential, and are now used for many practical applications with integration and exchange in relational databases [1, 8, 12].

But what about databases with more structure than a relational database, for example, XML documents [14]? Unlike a relational database, which is just a collection of tables, an XML document is ordered in a tree-like fashion. There has been some early work on incomplete information in XML [3, 10]. The paper [4] aims to provide a classification of problems associated with incompleteness for XML documents, like [2] and [9] did for relational databases.

Elements that can appear in XML documents are defined with DTDs [14]. A DTD (document type definition) is a set of declarations (a kind of a grammar) that define which elements may appear in an XML document, and how they are related. For example, consider a DTD for a database which describes the structure of employment in a company. Such a `company` might form a `group`; the description of each group might start with a `name` or not, followed by a description of a `leader` or not, followed by descriptions of `persons` employed in this group and smaller `groups` which are parts of the given group. The description of a `leader` consists of a description of a `person`. The description of a `person` consist of a `name`. The XML document below would be consistent with that DTD.

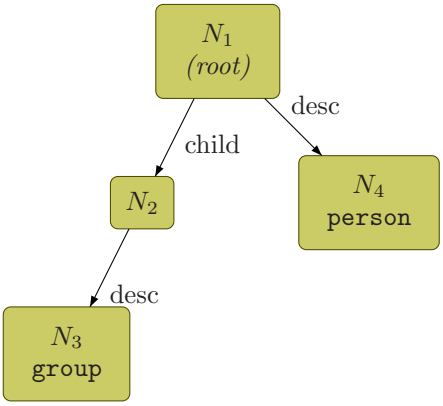
* Supported by the *Querying and Managing Navigational Databases* project realized within the Homing Plus programme of the Foundation for Polish Science, cofinanced by the European Union from the Regional Development Fund within the Operational Programme Innovative Economy ("Grants for Innovation").

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 367–380

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

```
<!DOCTYPE company [...]>
<company>
  <group>
    <name>Example Co.</name>
    <leader>
      <person><name>Smith</name></person>
      </leader>
    <group>
      <name>sales</name>
      <leader>
        <person><name>Baker</name></person>
        </leader>
      <person><name>Jones</name></person>
      </group>
    <group>
      <name>research</name>
      <person><name>Black</name></person>
      <person><name>White</name></person>
      </group>
    </group>
  </company>
```

An incomplete description of such a document might, for example, state that our document X contains four distinct nodes N_1, N_2, N_3, N_4 such that N_1 is the root, N_2 is a child of N_1, N_3 is a descendant of N_2, N_4 is a descendant of N_1, and N_3 is a `group` and N_4 is a `person`. The document above is consistent with this incomplete description (take N_1 to be the description of the company, N_2 to be the main group, N_3 to be the *research* group, and N_4 to be Black).



Note that this information is incomplete on several levels. It is possible that there are nodes about which we have no information at all (*open world assumption*). We don't know how many levels are between N_2 and N_3. In the assignment above, N_4 (Black) was a child of N_3 (research), although it was not explicitly stated in the description. In general, it would be also possible that N_2 could be a descendant of N_4—although our DTD forbids this, since a `person` cannot contain a `group` as a descendant.

[4] considers several types of such incomplete tree descriptions; we are interested in *incomplete DOM-trees* which enforce the mapping from the incomplete tree description to the tree to be injective, i.e., N_4 we know that is not equal to N_2 or N_3. Four *axes* are allowed to appear in the incomplete description: *next sibling*, *sibling*, *child*, *descendant*.

One of the problems investigated by [4] is the following:

▶ Problem 1.1. Let D be a fixed DTD. Given an incomplete tree description t, is there an XML document X which is consistent with both D and t?

Theorem 5.28 from [4] shows that this consistency problem is in PTIME for incomplete tree descriptions which do not use the *descendant* axis (only *child*, *sibling*, and *next sibling*). The case whether we can extend the result to also allow tree descriptions using this relation has been left open. Even the case where the tree description allows the *descendant* relation only on the topmost level and the shape of the descendant trees was completely fixed was an open question.

In this paper, we show how to solve this special case, and then show a polynomial algorithm for almost solving Problem 1.1. We say "almost", because there is a subtle difference between our and [4] understanding of when a tree is consistent with an incomplete description. However, we believe that our definition is also well motivated: although there are trees matching a given description according to [4] and not matching according to us, they are quite unnatural.

However, we prefer to work with theoretically more pure notions, rather than XML documents. Thus, instead of XML, we deal with binary trees with vertices labelled by elements of alphabet Σ, and instead of DTDs we deal with finite automata. The second change makes our results more general (each DTD corresponds to a finite automaton, but automata are more general than DTDs); another generalization is that we don't use the specific four axes listed above, but rather allow them to be defined using a regular expression. Still, we need to choose the axes from a fixed language $\mathcal{L}$ to obtain good algorithmic results. Our generalization from XML to generic trees also allows us to use our results for hierarchical data other than XML documents; for example, we could want to know whether a correct program in some programming language (or, more generally, a word in a context free language) exists that includes given keywords and symbols in given structural relationships; or whether there is an evolution of a branching process which exhibits given behaviors.

The paper is structured as follows. In Section 2 we provide the definitions required to understand the problem. In Section 3 we show how to solve the special case above in polynomial time (Theorem 3.2). This is used to explain techniques which are then used in Section 4 to solve the general problem for any tree descriptions (Theorem 4.3). In section 5 we show how to translate [4]'s problem from the XML world to the world of automata over binary trees, and point out the subtle difference that could not be solved with our methods.

For completeness, in Section 6 we show that the assumptions about the fixed size of automaton and the fixed set of languages cannot be lifted from Theorems 3.2 and 4.3. The problems become NP complete without these assumptions.

2 Preliminaries

An *unlabeled tree* is a finite $\tau \subset \{0,1\}^*$ such that whenever $uw \in \tau$, also $u \in \tau$. The empty word ϵ is the *root* of the tree, $w0$ and $w1$ are the *children* of w, and $w \in \tau$ is a *leaf* iff $w0, w1 \notin \tau$.

For a tree τ, by $\mathrm{port}(\tau)$ we denote the set of $x \in \{0,1\}^*$ such that $x \notin \tau$ and all proper prefixes of x are in τ. In other words, $\mathrm{port}(\tau)$ is the set of x such that $\tau \cup \{x\}$ is also a tree. The elements of $\mathrm{port}(\tau)$ are called *ports*; this name emphasizes our open world assumption – we don't treat these locations as places where the tree ends, but rather as variables: the tree can be extended ("grown") in arbitrary way from each port. In other words, our trees are similar to contexts used in the algebraic theory of trees ([5]), except that ports appear in every applicable location, not in a single one.

A *tree over* Σ is a function $T : \tau \to \Sigma$, where τ is an unlabeled tree. By $\mathrm{port}(T)$ we denote $\mathrm{port}(\mathrm{dom}(T))$.

Let T and U be two trees over Σ. We say that T is an *extension* of U ($T \subseteq U$) iff for each $u \in \mathrm{dom}(U)$ we have $u \in \mathrm{dom}(T)$ and $T(u) = U(u)$. We say that U is a *full subtree* of T at offset w iff for each u we have that $u \in \mathrm{dom}(U)$ iff $wu \in \mathrm{dom}(T)$, and $U(u) = T(wu)$. We say that U is an *inner subtree* of T at offset w iff an extension of U is a full subtree of T at offset w. If $U_1, U_2, \ldots, U_n$ are inner subtrees of T at offsets $w_1, w_2, \ldots, w_N$ respectively, we say that they are *disjoint inner subtrees* of T iff they share no common node, i.e., the sets $w_i\mathrm{dom}(U_i)$ are disjoint.

A (nondeterministic tree) *automaton* is a tuple $\mathcal{A} = (\Sigma, Q, q_I, \delta)$, where $q_0 \in Q$ and $\delta \subseteq \Sigma \times Q \times Q \times Q$.

Let T be a tree over Σ, and $\mathcal{A} = (\Sigma, Q, q_I, \delta)$ be an automaton. A *run* of $\mathcal{A}$ over T is a tree ρ over Q such that:

- $\mathrm{dom}(\rho) = \mathrm{dom}(T) \cup \mathrm{port}(T)$,
- For each $w \in \mathrm{dom}(T)$ we have $(T(w), \rho(w), \rho(w0), \rho(w1)) \in \delta$.

A run is *accepting* iff $\rho(\epsilon) = q_I$. We say that a tree T is *accepted* by $\mathcal{A}$ iff there is an accepting run of $\mathcal{A}$ on T. Sets of all trees accepted by some automaton $\mathcal{A}$ are called *regular languages* of trees, and can be equivalently defined using different kinds of automata, logic, algebra, etc.

This definition is a bit different from the usual one (e.g. [6]): in the usual definition, we would have a set of final states F, and force $\rho(x) \in F$ for each $x \in \mathrm{port}(T)$. As mentioned above, in our intentions (open world assumption) ports are places where the tree can extended in any way. If we want to make sure that our "incomplete" trees can be extended to trees which are accepted according to the usual definition, it is enough to assume that each state of $\mathcal{A}$ is productive, i.e., there is an accepting run on some tree which uses this state. Also, if we want to block some port x so that the tree cannot be extended there and the run ρ ends in a final state in this port, it is enough to add one extra character EOT to Σ and one extra state q_{EOT} to Q, and a transition $(\mathrm{EOT}, q, q_{\mathrm{EOT}}, q_{\mathrm{EOT}})$ for each $q \in F$. By putting $T(x) = \mathrm{EOT}$ we ensure that $\rho(x) \in F$ and $\rho(x0) = \rho(x1) = q_{\mathrm{EOT}}$, and since there is no transition with q_{EOT} on top, we can no longer extend the tree at $x0$ or $x1$ without losing acceptance. This allows us to simulate the usual definition with ours.

We will also use trees in their graph theoretical meaning. A *rooted tree* is a structure (V, E, v_R), where $E \subseteq V \times V$ and $v_R \in V$, such that for each $v \in V$ there is exactly one path from v_R to v in the graph (V, E). (This path is trivial for $v = v_R$.) By vE we denote the set of vertices $w \in V$ such that $(v, w) \in E$.

3 Special case

Let $\mathcal{A} = (\Sigma, Q, q_I, \delta)$ be a fixed automaton on binary trees over alphabet Σ.

▶ Problem 3.1. given trees $U_1, \ldots, U_N$. Decide whether there is a tree T accepted by $\mathcal{A}$ which includes $U_1, \ldots, U_N$ as disjoint inner subtrees, and one of the trees appears at offset ϵ.

The assumption that one of the trees appears at offset ϵ is to simplify the presentation by eliminating some of the special cases connected with the root. If we don't want such an assumption, it is enough to add an empty tree to our sequence.

Note that although the trees are disjoint, it is possible that e.g. U_2 will be connected to a port of U_3. It is also possible that U_2 is not directly connected to a port of U_3, but there is a path of vertices in between which do not belong to any U_i.

▶ **Theorem 3.2.** *Assuming that $|Q|$ is fixed, the problem 3.1 above can be solved in time polynomial in $|\Sigma|$ and the size of trees $U_1, \ldots, U_N$.*

Proposition 6.1 in Section 6 below shows that the problem becomes NP complete without the assumption that $|Q|$ is fixed.

We will present the proof of the theorem above in the following way. We will show a sequence of simpler algorithms; for each of them, we will identify the major problem with it and show how to fix it. This sequence will converge to a correct algorithm running in deterministic polynomial time.

We will start with some additional definitions.

3.1 Multiplicity vectors

We call functions $f : Q \to \mathbb{Z}$ *multiplicity vectors*. For $x \in \mathbb{Z}$, we say $f \geq x$ iff $f(q) \geq x$ for all $q \in Q$; $f \leq x$ is defined similarly. By $[q]$ we denote the multiplicity vector such that $[q](q) = 1, [q](r) = 0$ for $r \neq q$. For a set of multiplicity vectors S, by $S[q > 0]$ we denote $\{f \in S : f(q) > 0\}$. If S_1 and S_2 are sets of multiplicity vectors, $S_1 + S_2 = \{f_1 + f_2 : f_1 \in S_1, f_2 \in S_2\}$, and $S_1 - S_2 = \{f_1 - f_2 : f_1 \in S_1, f_2 \in S_2\}$.

We say that a tree T **realizes** a multiplicity vector f from $q \in Q$ iff $f \geq 0$ and there is a valid run ρ of $\mathcal{A}$ over T such that for each state r, r appears in the ports of T at least $f(r)$ times (i.e., the cardinality of $\rho^{-1}(r) \cap \mathrm{port}(T)$ is at least $f(r)$), and $\rho(\epsilon) = q$.

We denote by $A(U, q, n)$ the set of multiplicity vectors f such that $f \leq n$ and there is a tree $T \supseteq U$ which realizes f from q.

▶ **Lemma 3.3.** *Sets $A(U, q, n)$ can be calculated in time polynomial in $|U|$, S, and n.*

Proof. We start with calculating $A(\emptyset, q, n)$ for each q.

Define the sequence of sets of multiplicity vectors $A_i(q, n)$ as follows:

- $A_0(q, n) = \{f : 0 \leq f \leq n, f \leq [q]\}$
- $A_{i+1}(q, n)$ is a set of f such that $0 \leq f \leq n$ and there exist $f_1 \in A_i(q_1, n)$ and $f_2 \in A_i(q_2, n)$ such that $(x, q, q_1, q_2) \in \delta$ and $f \leq f_1 + f_2$.

▶ Proposition 3.4. If $A_i(q, n) = A_{i+1}(q, n)$ for each $q \in Q$, then $A(\emptyset, q, n) = A_i(q, n)$.

The proof is straightforward. This proposition allows us to calculate $A(\emptyset, q, n)$ for each q and n inductively.

▶ Proposition 3.5. Let U be a tree such that U_1 and U_2 are full subtrees of U at offsets 0 and 1, respectively. Then $A(U, q, n)$ is the set of f such that there exist $f_i \in A(U_i, q_i, n)$ for $i = 1, 2$ such that $(U(\epsilon), q, q_1, q_2) \in \delta$ and $f \leq f_1 + f_2$.

Again, the proof is straightforward. This proposition allows us to calculate $A(U, q, n)$ by induction over subtrees of U. ◀

3.2 Algorithm I

We start with a strongly non-deterministic algorithm.

1. We guess a permutation of our set of trees: $U_{d_1}, U_{d_2}, U_{d_3}, \ldots, U_{d_N}$. This specifies the order of subtrees in our goal tree T. If U_i appears at offset w_i in the goal tree T, and w_i is a prefix of w_j, then the tree U_i will appear before U_j in this permutation. If w_i and w_j are incomparable, they can be ordered arbitrarily.
2. $f := [q_I]$. At each stage of our algorithm, f is the multiplicity vector which says how many occurrences of each state we have in our ports.
3. for $i := 1$ to N:

4. We guess $q \in Q$.
5. If $f(q) = 0$, then we fail.
6. $f := f - [q]$
7. $f := f + a$, where a is one of the elements of $A(U_{d_i}, q, N)$.
8. We accept.

In our loop, we fill one of our ports in some state q with the tree U_{d_i}. This means we used one of the ports specified by $f(q)$, but U_{d_i} gives us new ports according to $A(U_{d_i}, q, N)$. It is enough to use multiplicity vectors f such that $f \leq N$ because we will only have to use N states.

3.3 Algorithm II

Except for Step 1, all the steps of Algorithm I are easy to determinize in polynomial time. Instead of analyzing one choice, we construct the sets of all possible choices.

1. We guess a permutation of our set of trees, just like in Algorithm I.
2. $S := \{[q_I]\}$
3. for $i := 1$ to N:
4. $\quad P := \emptyset$;
5. $\quad$ for $q \in Q$ do
6. $\quad\quad R := S$;
7. $\quad\quad R := R[q > 0]$; // use only vectors which had a port with state q
8. $\quad\quad R := R - [q]$; // fill this port
9. $\quad\quad R := R + A(U_{d_i}, q)$; // we get new states in ports below U_{d_i}
10. $\quad\quad P := P \cup R$;
11. $\quad$ S := P;
12. We accept.

If in the end we have obtained a non-empty set S, we have won (constructed a tree which contains our required subtrees). If not, it probably means that we have guessed the permutation incorrectly.

Now, our goal is to avoid guessing the permutation.

3.4 Algorithm III

In Algorithm III we do not guess the permutation. Instead, we do not care if we have used up a state which we have not obtained yet — i.e., we remove Step 7 from Algorithm II, and allow our multiplicity vectors to go negative.

At the end of our algorithm we check whether S has an element $f \geq 0$. If no, it means that we had no chance – whatever permutation we would pick in Algorithm II or Algorithm I, we would get an empty set at the end. What if S contains a non-negative element? This does not yet mean that the tree exists, because it is still possible that it was impossible to do a good ordering. For example, suppose we have two states q and r which are unreachable from q_I, U_1 is an empty tree, and the tree U_2 realizes $[q]$ from r and U_3 realizes $[r]$ from q. Algorithm III will accept, even if the trees U_2 and U_3 can only be used in states which are not reachable from q_I.

3.5 Algorithm IV

Thus, we need to make sure that a correct permutation exists. For this, we use the following lemma:

▶ **Lemma 3.6.** *For $i = 1 \dots N$, let $f_i \in A(U_i, q_i, N)$. Then a permutation of trees for the Algorithm I which generates a positive answer, such that in step 3 for each i we choose to use state q_{d_i} as q and multiplicity vector f_{d_i} as a, exists iff the following two conditions are satisfied:*

- *Euler condition:* $[q_I] + \sum(f_i - [q_i]) \geq 0$
- *Connectedness: Let $G = (V, E)$ be the graph such that $V = \{q_i : i \in \{1 \dots n\}\}$, and $(p, q) \in E$ iff there exists an i such that $p = q_i$ and $f_i(q) > 0$. Then there must be a path from q_I to each state in V. edge $p \to q$. Then there must be a path from q_I to each state in V.*

This lemma is a generalization of the classic Euler theorem about a condition for existence of an Euler path in a graph. Also the proof generalizes the proof of Euler's result. This technique is essentially equivalent to Theorem 3.1 from [7] and Proposition 3 from [11].

Proof. We try to build the tree in an arbitrary way, filling up ports with our trees whenever the state matches. If we end up with all trees used, we are done. Otherwise, there is some tree U from r_0 which cannot be inserted because all ports of type r_0 have been already used. We can assume that r_0 was already used as a connecting state in our construction; otherwise, there would be no connection between the used and unused trees, which means that the connectedness condition is not satisfied. We construct a sequence of states $r_0, r_1, r_2, \dots, r_m$ and of unused trees $U_{i_0}, U_{i_1}, \dots$, such that $q_{i_k} = r_{k+1}$ and $f_{i_k}(r_k) > 0$, until we no longer can find an unused tree U_{i_m} with $f_{i_m}(r_m) > 0$. This construction must end in the state $r_m = r_0$; if it ended in any other state, it would violate the Euler condition (the state r_m would be used in the root more times that it would be produced in a port). We find the place in our construction where r_0 is used for connection, and insert the trees $U_{i_{m-1}}, \dots, U_{i_0}$ there. We continue this operation until all trees are used. ◀

Algorithm III checked for the Euler condition, but we have to modify it to also check for connectedness. This can be done by choosing the (connected) graph G, or even better, guessing the spanning tree of G. Since $V \subseteq Q$ and $|Q|$ is fixed, we have a fixed number of possible spanning trees. Thus, we modify Algorithm III in the following way:

1. We start by guessing a rooted tree $\tau = (V, E, q_I)$. (A fixed number of possible guesses.)
2. Now, for each edge $e \in E$ we guess i_e, the index of the tree which is forced to realize this edge. (Thus, we have $N^{|V|-1}$ possible guesses, which again is polynomial.)
3. We execute Algorithm III, except that in step 5, if $i = i_{(q_1,q_2)}$ for some $e \in E$, then we use only q_1 as q, and in general we can only use states from V; and in step 9 we restrict $A(t_{d_i}, q, n)$ to multiplicity vectors f such that $f(q_2) > 0$.

Algorithm IV is correct and works in deterministic polynomial time (when guesses are replaced by looping over the whole subset).

4 Incomplete tree descriptions

In this section we generalize the results from the section below to a case where we can force the trees U_i to appear in the result tree in a specific pattern.

As in the previous section, let $\mathcal{A} = (\Sigma, Q, q_I, \delta)$ be a fixed tree automaton over Σ. Also let let $\mathcal{L}$ be a finite set of regular (word) languages over $\{0, 1\}$.

An **incomplete tree description**, or a **pattern** for short, is a $P = (V^P, E^P, p_0, C, L)$, where:

- (V^P, E^P, p_0) is a rooted tree,
- $C : V^P \to P(\Sigma)$ assigns a set of possible elements of alphabet to each subpattern,
- $L : E^P \to \mathcal{L}$ assigns one of the languages in $\mathcal{L}$ to each edge of our tree.

For $p \in V^P$, $P[p]$ is the subpattern obtained from P by moving the root p_0 to p and restricting V^P to vertices accessible from p.

▶ **Definition 4.1.** A tree T **matches** an incomplete tree description P iff $T(\epsilon) \in c(P)$, and for each $p \in p_0 E^P$, a tree U_p matching $P[p]$ is an inner subtree of T at offset $w_p \neq \epsilon$, and they are disjoint inner subtrees.

▶ Problem 4.2. Given an incomplete tree description P, is there a tree T accepted by $\mathcal{A}$ which matches P?

Note that the set of trees which matches a given incomplete description P is a regular language of trees; and an intersection of two regular languages is also a regular language. However, an automaton recognizing trees consistent with P would be of size exponential in $|P|$, so such a view is not practical for us.

▶ **Theorem 4.3.** *Let $\mathcal{A}$ and $\mathcal{L}$ be fixed. Problem 4.2 is solvable in time polynomial in the size of P.*

Propositions 6.1 and 6.2 in Section 6 below show that the problem becomes NP complete without the assumption that $\mathcal{A}$ and $\mathcal{L}$ are fixed, respectively.

Note that the Problem 3.1 is the special case of Problem 4.2 where $\mathcal{L} = \{0, 1, (0 + 1)^*\}$ and $(0 + 1)^*$ is only allowed to appear at edges starting in p_0 (not counting some minor differences regarding the root of the result tree).

Proof. For words $u, v \in \{0, 1\}^*$, we say that $u \equiv v$ iff for each two words $t, w \in \{0, 1\}^*$ and each $L \in \mathcal{L}$ we have $tuw \in L$ iff $tvw \in L$. It is well known that the relation $\equiv$ has a finite index. Let M be the set of equivalence classes of $\equiv$; let $[w] \in M$ be the equivalence class of a word $w \in \{0, 1\}^*$. M is equipped with a concatenation operation given by $[w_1][w_2] = [w_1 w_2]$. For a $L \in \mathcal{L}$, let $[L] = \{[w] : w \in L\}$; note that for each $m \in M$ either $m \subseteq L$ or m is disjoint with L. The set M is called the *syntactic monoid* of $\mathcal{L}$ (see e.g. [13]; although syntactic monoids are more commonly known for single languages, our extension of this notion to a finite family of languages is quite obvious).

Let P be an incomplete tree description, and $N = |P|$.

We will need to extend our definition of multiplicity vectors to take the elements of M into account together with states. An **extended multiplicity vector** (EMV) is a function $f : Q \times M \to \mathbb{Z}$. We say that a tree T realizes an EMV f from $q \in Q$ iff there is a valid run ρ of $\mathcal{A}$ over T such that $\rho(\epsilon) = q$, and for each $r \in Q$ and $m \in M$, $\left|\rho^{-1}(r) \cap \mathrm{port}(T) \cap m\right| \geq f(r, m)$. We define all operations on EMVs and sets of EMVs in the same way as for multiplicity vectors.

If f is an EMV and $m \in M$ we denote by mf the EMV such that $mf(q, m_1) = \sum_{m_2 : mm_2 = m_1} f(q, m_2)$.

For $p \in V_P$ and a state $q \in Q$, we denote by $A(p, q, n)$ the set of EMVs f such that $0 \leq f \leq n$, and there is a tree U which matches $P[p]$ and realizes f from q.

Now, it is enough to show that $A(p_0, q, n) \neq \emptyset$. Thus, all we need is the following lemma:

▶ **Lemma 4.4.** *The set $A(p, q, n)$, where $n \leq N$, can be calculated in polynomial time for each $p \in V_P$.*

Thus, to check whether a tree accepted by $\mathcal{A}$ and matching P exists, it is enough to check whether $A(p_0, q_I, 0) \neq \emptyset$. ◀

To prove 4.4 we will need one more technical lemma:

▶ **Lemma 4.5.** *Let $q \in Q$ and $x \in \Sigma$. Let set $A^0(x, q, n)$ be the set of EMVs f which are realized from q by a tree which has x in its root, and $0 \leq f \leq n$. The set $A^0(x, q, n)$ (where $n \leq N$) can then be calculated in polynomial time.*

Proof. Let $A^0_0(x, q, n)$ be the set of EMVs which can be realized by a tree which has x in its root and ports in both of its children.

Let $A^0_{k+1}(x, q, n)$ be the set of EMVs f such that either $f \in A^0_0(x, q, n)$, or the following conditions are satisfied for some EMVs f_0 and f_1, states $q_0, q_1 \in Q$, and letters $x_0, x_1 \in \Sigma$:

- $f \leq f_0 + f_1$
- $f_0 \in [0]A^0_k(x_0, q_0, n)$
- $f_1 \in [1]A^1_k(x_1, q_1, n)$
- $0 \leq f \leq n$

It is straightforward to check that the sequence $A^0_k(x, q, n)$ is increasing for each x, q, n, and its limit is $A^0(x, q, n)$. ◀

Proof of Lemma 4.4. We prove the lemma by induction over subpatterns. Let $n' = n + |pE_P|$. From inductive hypothesis we can calculate the $A(p', r, n)$ for each $p' \in pE_P$ and each $r \in Q$.

1. $A := \emptyset$
2. for each rooted tree $\tau = (V^\tau, E^\tau, v^\tau)$ such that $V^\tau \subseteq Q \times M$ and $v^\tau = (q, [\epsilon])$:
3. for each assignment $\alpha : E^\tau \to pE_P \cup \{p\}$:
4. $\quad B := \bigcup_{x \in C(P)} A^0(x, q, n')$
5. $\quad \mathrm{Restrict}_\alpha(B, (q, [\epsilon]), p)$
6. $\quad$ For each $p' \in pE_P$:
7. $\quad\quad$ For each $(r, m) \in Q \times [L(p, p')] \cap V^\tau$:
8. $\quad\quad\quad A := mA(p', r, n')$
9. $\quad\quad\quad \mathrm{Restrict}_\alpha(A, (r, m), p')$
10. $\quad\quad\quad$ let $B := B - [(r, m)] + A$
11. $\quad$ Let $A := A \cup \{f \in B : 0 \leq f \leq n\}$
12. return A

$\mathrm{Restrict}_\alpha(B, (q, m), p)$ is defined as follows:

1. For each $((q_1, m_1), (q_2, m_2)) \in \alpha^{-1}(p)$:
2. $\quad$ if $(q_1, m_1) \neq (q, m)$ then $B := \emptyset$
3. $\quad B := B[(q_2, m_2) > 0$

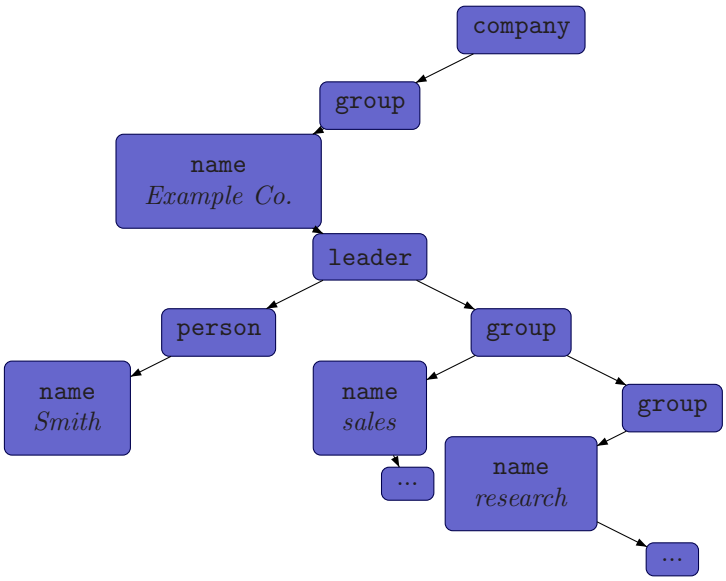
This algorithm is based on a idea which is very similar to that of Algorithm IV. The differences is that it takes the syntactic monoid M into account, and calculates the EMV instead of non-emptiness. ◀

5 Application to XML

In this section, we compare the problem solved in Theorem 4.3 to the open question from [4]: is it possible to remove the condition of $\downarrow^*$-freeness from Theorem 5.28? In other words, for a fixed DTD D, and an incomplete DOM-tree t, is it possible to find out whether there exists an XML document X which satisfies D and is consistent with t, in time polynomial in size of t?

It is beyond scope of this paper to include the full definition of XML documents, DTDs, and incomplete DOM trees, thus we just give examples and list the important differences:

- XML documents are trees, but they are unranked trees (with a sibling ordering), not binary ones. However, we can use the standard encoding of unranked trees in binary trees: if a vertex v of the unranked tree is assigned w in our binary tree, then the first child of v (if any) is assigned $w0$, and the next sibling of v (if any) is assigned $w1$. This allows us to convert any unranked tree (or, more accurately, an unranked forest) to a binary tree, and vice versa. The following picture shows the XML document from the introduction, represented as a binary tree.

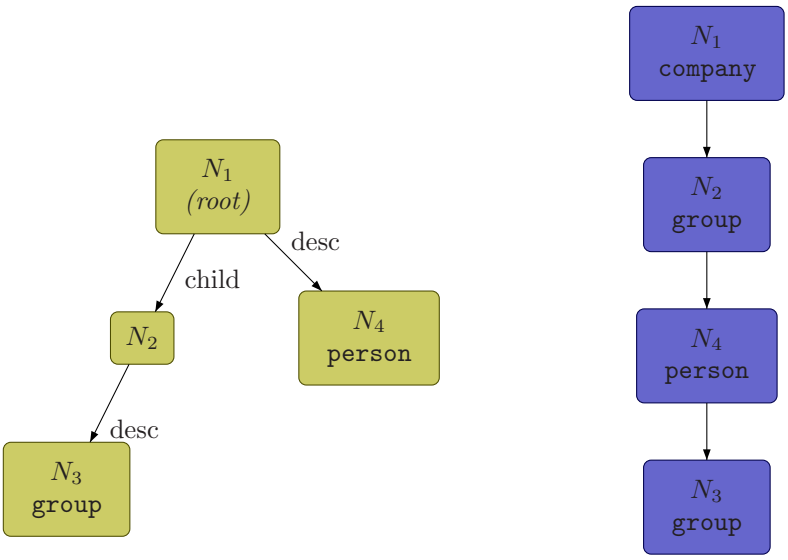


- In [4], the question is whether a tree consistent with a given DTD exists. Essentially, for each type of a node, a DTD (document type definition) gives a regular expression which describes which sequences of node types can be allowed as children of a node of given type. For example, we can use the following DTD to define valid company descriptions:

```
<!ELEMENT company (group)>
<!ELEMENT group (name? leader? (group|person)*)>
<!ELEMENT leader (person)>
<!ELEMENT person (name)>
<!ELEMENT name (#PCDATA)>
```

In our results, we use tree automata instead of DTDs. Our result is more general, since tree automata can verify whether a tree is consistent with a DTD, but not the other way around – tree automata are more powerful. Properties such as *if a group has a leader and only one subgroup, then this subgroup cannot have a leader* cannot be easily described with a DTD, but can be described with an automaton.

- Incomplete DOM-trees allow using markings: in terms of patterns, each subpattern can have marking which forces it to appear in a specific location, which is one of the following: *root, leaf, first child, last child.* These markings can be easily checked by extending the automaton $\mathcal{A}$ and the alphabet Σ, or by enforcing EOT on the applicable child, so our result covers this.
- Subpatterns are allowed to have only one of the following relations to their bigger nodes: *next sibling*, *younger sibling*, *child*, *descendant.* This corresponds to picking a language 1, 1^*, 01^*, $0(0+1)^*$, respectively, as $L(p, p')$. Thus, our result again generalizes the XML case.
- The last difference cannot be easily solved with our means. In case of [4], a tree T is considered to match pattern P if there is an injective mapping ϕ from V_P to vertices of T such that for each edge $(p_1, p_2) \in E_P$ we have $\phi(p_2) = \phi(p_1)v$ where $v \in L(p_1, p_2)$. There is a subtle difference with our definition. Consider again the pattern from the introduction (on the left), and the following tree (on the right):



Is the tree to the right consistent with the pattern on the left? According to the definition from [4], yes ($\phi(v)$ is the node of the tree which is labelled with the same N_i as v). According to our definition, no: the inner subtree which matches the subpattern rooted at N_2 would have to include N_4, which means that it would not be disjoint with the inner subtree which matches the subpattern N_4. This is the only case where there is a difference: if N_4 is below N_3, above N_2, between N_2 and N_3, or below N_2 but not above N_3, then the tree matches the description according to both definitions.

In general the situation can be more complicated (there could be long sequences of descendants which the definition from [4] would allow to interleave in arbitrary ways). We don't see how to use inductive reasoning to exactly solve the case from [4]. However, we think that it is not natural to interleave vertices which correspond to different subpatterns, and for this reason both definitions are motivated equally well.

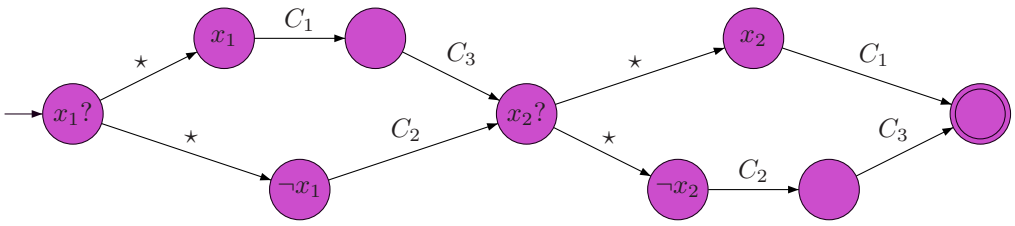
6 Hardness results

In this section we show that the assumptions about the fixed size of Q and $\mathcal{L}$ cannot be removed from Theorems 3.2 and 4.3.

▶ Proposition 6.1. Without assumption that $|Q|$ is fixed, Problem 1 is NP complete, even in the word case, and when all tress U_i contain only one letter.

This has been essentially proven in [4]; we provide a short proof for completeness.

Proof. The problem is in NP because the witness must be polynomial. We reduce the problem of CNF-SAT satisfiability to Problem 1. Let $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_n$ be a CNF formula, with variables $x_1, \ldots, x_m$. Our language will be $\Sigma = \{\star, C_1, \ldots, C_n\}$. For each x_i our automaton contains two paths, one for x_i and one for $\neg x_i$; on each path we accept a letter corresponding to all clauses that are satisfied by choosing given literal. Tree U_i asks whether C_i appears anywhere in the resulting word. The following picture shows the automaton used to decide whether a formula $\phi = C_1 \wedge C_2 \wedge C_3$ is satisfiable, where $C_1 = x_1 \vee x_2$, $C_2 = \neg x_1 \vee \neg x_2$, and $C_3 = x_1 \vee \neg x_2$.



◀

Note that we could get a more straightforward reduction from the Hamiltonian circuit problem if we could say that the result tree T is covered by U_is completely. After a small modification, our technique allows to solve Problems 3.1 and 4.2 even in this case (or in the case where we have a subset $\Delta \subseteq \Sigma$ and labels from Δ are only allowed to appear when explicitly requested by the description).

Also note that we could encode the reduction from the proof of Proposition 6.1 in a two letter alphabet, but then the trees (or word) U_i need to be longer. Restricting to both an alphabet of fixed size and trees U_i of size 1 again yields a polynomial algorithm (which is similar to one given in Lemma 3.3, but we count the number of occurrences of each label, not each state; a similar problem has been solved in [11]).

▶ Proposition 6.2. Without assumption that $\mathcal{L}$ is fixed, Problem 2 is NP complete, even for a very simple (fixed) automaton ($|\sigma| = 1, |Q| = 2, |\delta| = 2$), and incomplete tree descriptions of depth 2 (i.e., consisting just of a root and its direct subpatterns).

Proof. Our automaton says that the tree consists of a single path. Thus, we have $\Sigma = \{a\}, Q = \{q_I, q_N\}, \delta = \{(a, q_I, q_I, q_N), (a, q_I, q_N, q_I)\}$.

Again, the problem is in NP because the witness must be polynomial, and we prove hardness by reducing CNF satisfiability. Let $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_n$ be a CNF formula, with variables $x_1, \ldots, x_m$. Let D_i be the set of sequences of m bits S such that C_i is satisfied when x_j is satisfied iff $S_j = 1$. The subpattern P_i says there is a vertex labeled with a at position $D_i\{0 + 1\}^*$; this regular language can be defined in polynomial size (if defined as a

regular expression or a DFA; the syntactic monoid is of exponential size, though). Since the tree must have only one path, all D_i parts much correspond to the same word, which means that all C_i are satisfiable simultaneously. ◀

We don't know whether the problem is already NP complete for a trivial automaton. However, if we use the injective definition of consistence with an incomplete description from [4], and a trivial automaton, then the problem is again NP complete in the word case. Indeed, we can reduce the 3-CNF satisfiability problem: for each of the m variables we have a subpattern p_i, and $L(p_0, p_i) = 1^{im} + 1^{im+i}$; putting the tree matching the subpattern at offset 1^{im} means that x_i is false, and offset 1^{im+i} means that x_i is true. Then, for each clause we add additional subpatterns to respective p_i's, so that if the three variables have been given the only assignment which does not satisfy the clause, we have to fit three different offsets into two slots (choosing any other assignment gives us more space). Note that the syntactic monoid M – even the one which recognizes all languages L at once – is still of polynomial size in this case.

7 Conclusion

We have shown how to determine in polynomial time whether there exists a tree which is accepted by the given automaton $\mathcal{A}$ and is consistent with given incomplete description P, assuming that both the automaton $\mathcal{A}$ and the set of possible axes in P are fixed. This brings us closer to a complete understanding of the consistence problem for XML documents [4] and other hierarchical data. Although our goal was to improve the results of [4], we had to use another definition of when a tree matches a incomplete description in order to make our method work. Our definition differs in just one subtle detail — while the original one allowed several branches of the incomplete description to be mixed together and appear as a single branch in the result tree, our does not allow that — they can become a single branch, but the tree matching one of the subpatterns must completely precede the tree matching other subpatterns, no mixing is allowed. We believe that such mixing is not natural, so our result is also interesting. Still, the original consistency problem, for the original definition, remains open.

One of techniques we have been using is Lemma 3.6, which is essentially the classical Euler theorem about existence of Euler path, generalized to the case where an "edge" can have many endpoints. This technique has been also used recently in [11] (and also less recently in [7]). We believe that it is worth investigating whether this method has more applications for various kinds of branching structures in automata theory and computer science in general.

Acknowledgements I want to thank Pablo Barcelo and Filip Murlak for introducing me to these problems and their helpful comments, and Wojtek Czerwiński and Paweł Parys for the atmosphere of research at AUTOBÓZ 2010.

References

1 Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '98, pages 254–263, New York, NY, USA, 1998. ACM.

2 Serge Abiteboul, Paris Kanellakis, and Gosta Grahne. On the representation and querying of sets of possible worlds. *SIGMOD Rec.*, 16:34–48, December 1987.

3 Serge Abiteboul, Luc Segoufin, and Victor Vianu. Representing and querying xml with incomplete information. *ACM Trans. Database Syst.*, 31:208–254, March 2006.

4 Pablo Barceló, Leonid Libkin, Antonella Poggi, and Cristina Sirangelo. Xml with incomplete information. *J. ACM*, 58:4:1–4:62, December 2010.

5 Mikolaj Bojańczyk and Igor Walukiewicz. Forest algebras. In *Logic and Automata'08*, pages 107–132, 2008.

6 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

7 Javier Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundam. Inf.*, 31:13–25, July 1997.

8 Ronald Fagin, Phokion Kolaitis, Renée Miller, and Lucian Popa. Data exchange: Semantics and query answering. In Diego Calvanese, Maurizio Lenzerini, and Rajeev Motwani, editors, *Database Theory - ICDT 2003*, volume 2572 of *Lecture Notes in Computer Science*, pages 207–224. Springer Berlin / Heidelberg, 2002. 10.1007/3-540-36285-1_14.

9 Tomasz Imieliński and Witold Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31:761–791, September 1984.

10 Yaron Kanza, Werner Nutt, and Yehoshua Sagiv. Querying incomplete information in semistructured data. *Journal of Computer and System Sciences*, 64(3):655 – 693, 2002.

11 Eryk Kopczynski and Anthony Widjaja To. Parikh images of grammars: Complexity and applications. In *Proceedings of the 2010 25th Annual IEEE Symposium on Logic in Computer Science*, LICS '10, pages 80–89, Washington, DC, USA, 2010. IEEE Computer Society.

12 Maurizio Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 233–246, New York, NY, USA, 2002. ACM.

13 Jean-Eric Pin. Syntactic semigroups. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of formal languages, vol. 1*, pages 679–746. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

14 W3C. Extensible markup language (xml) 1.1 (second edition). Available on: `http://www.w3.org/TR/xml11/`, 2006.

A Formal Theory for the Complexity Class Associated with the Stable Marriage Problem

Dai Tri Man Lê, Stephen A. Cook, and Yuli Ye

Department of Computer Science, University of Toronto

Abstract

Subramanian defined the complexity class CC as the set of problems log-space reducible to the comparator circuit value problem. He proved that several other problems are complete for CC, including the stable marriage problem, and finding the lexicographical first maximal matching in a bipartite graph. We suggest alternative definitions of CC based on different reducibilities and introduce a two-sorted theory VCC^* based on one of them. We sharpen and simplify Subramanian's completeness proofs for the above two problems and formalize them in VCC^*.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems; F.4.1 Mathematical logic

Keywords and phrases Bounded arithmetic, complexity theory, comparator circuits

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.381

1 Introduction

Comparator networks were originally introduced as a method of sorting numbers (as in Batcher's even-odd merge sort [2]), but they are still interesting when the numbers are restricted to the Boolean values $\{0, 1\}$. A comparator gate has two inputs p, q and two outputs p', q', where $p' = \min\{p, q\}$ and $q' = \max\{p, q\}$. In the Boolean case (which is the one we consider) $p' = p \wedge q$ and $q' = p \vee q$. A comparator circuit (i.e. network) is presented as a set of m horizontal lines in which the m inputs are presented at the left ends of the lines and the m outputs are presented at the right ends of the lines, and in between there is a sequence of comparator gates, each represented as a vertical arrow connecting some wire w_i with some wire w_j as shown in Fig. 1. These arrows divide each wire into segments, each of which gets a Boolean value. The values of wires w_i and w_j after the arrow are the comparator outputs of the values of wires w_i and w_j right before the arrow, with the tip of the arrow representing the maximum.

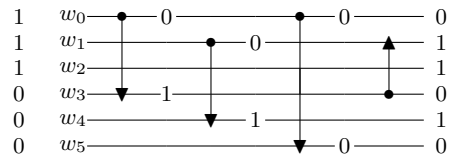


Figure 1

The comparator circuit value problem (Ccv) is: given a comparator circuit with specified Boolean inputs, determine the output value of a designated wire. To turn this into a complexity class it seems natural to use a reducibility notion that is weak but fairly robust. Thus we define CC to consist of those problems (uniform) AC^0 many-one-reducible to Ccv. However Subramanian [9] studied the complexity of Ccv using a stronger notion of reducibility. Thus his class, which we denote $\mathsf{CC}^{\mathsf{Subr}}$, consists of those problems log-space (many-one)-reducible to Ccv. It turns out that a generalization of AC^0 many-one reducibility which we will call AC^0 *oracle reducibility* (called simply AC^0 reducibility in [3]), is also useful. Standard complexity classes such as AC^0, L (log space), NL (nondeterministic log space), NC, and P are all closed under this AC^0 oracle reducibility. We denote the closure of Ccv under this reducibility by CC^*.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 381–395
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We will show that

$$\mathsf{NL} \subseteq \mathsf{CC} \subseteq \mathsf{CC}^{\mathsf{Subr}} \subseteq \mathsf{CC}^* \subseteq \mathsf{P} \tag{1.1}$$

The last inclusion is obvious because Ccv is a special case of the monotone circuit value problem, which is clearly in P. The inclusion $\mathsf{CC} \subseteq \mathsf{CC}^{\mathsf{Subr}}$ follows because AC^0 functions are also log-space functions. The inclusion $\mathsf{CC}^{\mathsf{Subr}} \subseteq \mathsf{CC}^*$ follows from the first inclusion, which in turn is a strengthening of a result in [6] (attributed to Feder) showing that $\mathsf{NL} \subseteq \mathsf{CC}^{\mathsf{Subr}}$. Of course all three comparator classes coincide if it turns out that CC is closed under AC^0 oracle reductions, but we do not know how to show this.

Note that comparator circuits are more restricted than monotone Boolean circuits because each comparator output has fan-out one. This leads to the open question of whether $\mathsf{CC}^* \subsetneq \mathsf{P}$. A second open question is whether CC^* and NC are incomparable. (Here NC is the class of problems computed by uniform circuit families of polynomial size and polylog depth, and satisfies $\mathsf{NL} \subseteq \mathsf{NC} \subseteq \mathsf{P}$.) The answers could be different if we replaced CC^* by $\mathsf{CC}^{\mathsf{Subr}}$ or CC, although $\mathsf{CC} \subseteq \mathsf{NC}$ iff $\mathsf{CC}^* \subseteq \mathsf{NC}$ because NC is closed under AC^0 oracle reductions.

The above classes associated with Ccv are also interesting because they have several disparate complete problems. As shown in [6, 9] both the lexicographical first maximal matching problem (Lfmm) and the stable marriage problem (Sm) are complete for $\mathsf{CC}^{\mathsf{Subr}}$ under log-space reductions[1]. The Sm problem is especially interesting: Introduced by Gale and Shapley in 1962 [4], it has since been used to pair medical interns with hospital residencies jobs in the USA. Sm can be stated as follows: Given n men and n women, each with a complete ranking according to preference of all n members of the opposite sex, find a complete matching of the men and women such that there are no two people of opposite sex who would both rather have each other than their current partners. Gale and Shapley proved that such a 'stable' matching always exists, although it may not be unique. Subramanian [9] showed that Sm treated as a search problem (i.e. find any stable marriage) is complete for CC under log-space reducibility.

Strangely the CC classes have received very little attention since Subramanian's papers [8, 9]. The present paper contributes to their complexity theory by sharpening these early results and simplifying their proofs. For example we prove that the three problems Ccv, Lfmm, and Sm are inter-reducible under AC^0 many-one reductions as opposed to log-space reductions. Also we introduce a three-valued logic version of Ccv to facilitate its reduction to Sm. Our paper contributes to the proof complexity of the classes by introducing a two-sorted formal theory VCC^* which captures the class CC^* and which can formalize the proofs of the above results.

Our theory VCC^* is a two-sorted theory developed in the way described in [3, Chapter 9]. In general this method associates a theory VC with a suitable complexity class C in such a way that a function is in FC, the function class associated with C, if and only if it is provably total in VC. (A string-valued function is in FC iff it is polynomially bounded and its bit-graph is in C.) This poses a problem for us because the provably-total functions in a theory are always closed under composition, but it is quite possible that neither of the function classes FCC and $\mathsf{FCC}^{\mathsf{Subr}}$ is closed under composition. That is why we define the class CC^* to consist of the problems AC^0-oracle-reducible (see Definition 3 below) to Ccv, rather than the problems AC^0 many-one reducible to Ccv, which comprise CC. It is easy to

[1] The second author outlined a proof that Lfmm is complete under NC^1 reductions in unpublished notes from 1983.

see that the functions in FCC^* are closed under composition, and these are the functions that are provably total in our theory VCC^*.

The above paragraph illustrates one way that studying proof complexity can contribute to main-stream complexity theory, namely by mandating the introduction of the more robust version CC^* of CC and $\mathsf{CC}^{\mathsf{Subr}}$. Another way is by using the simple two-sorted syntax of our theories to demonstrate AC^0 reductions. Thus Theorem 1 below states that a simple syntactic class of formulas represents precisely the AC^0 relations. In general it is much easier to write down an appropriate such formula than to describe a uniform circuit family or alternating Turing machine program.

Once we describe our theory VCC^* in Sections 2.1 and 2.2, the technical part of our proofs involve high-level descriptions of comparator circuits and algorithms. We do not say much about formalizing the proofs in VCC^* since this part is relatively straightforward.

2 Preliminaries

2.1 Two-sorted vocabularies

We use two-sorted vocabularies for our theories as described by Cook and Nguyen [3]. Two-sorted languages have variables $x, y, z, \ldots$ ranging over $\mathbb{N}$ and variables $X, Y, Z, \ldots$ ranging over finite subsets of $\mathbb{N}$, interpreted as bit strings. Two sorted vocabulary $\mathcal{L}_A^2$ includes the usual symbols $0, 1, +, \cdot, =, \leq$ for arithmetic over $\mathbb{N}$, the length function $|X|$ for strings ($|X|$ is zero if X is empty, otherwise $1 + \max(X)$), the set membership relation $\in$, and string equality $=_2$ (subscript 2 is usually omitted). We will use the notation $X(t)$ for $t \in X$, and think of $X(t)$ as the t^{th} bit in the string X.

The number terms in the base language $\mathcal{L}_A^2$ are built from the constants $0, 1$, variables $x, y, z, \ldots$ and length terms $|X|$ using $+$ and $\cdot$. The only string terms are string variables, but when we extend $\mathcal{L}_A^2$ by adding string-valued functions, other string terms will be built as usual. The atomic formulas are $t = u$, $X = Y$, $t \leq u$, $t \in X$ for any number terms x, y and string variables X, Y. Formulas are built from atomic formulas using $\wedge, \vee, \neg$ and $\exists x$, $\exists X$, $\forall x$, $\forall X$. Bounded number quantifiers are defined as usual, and bounded string quantifier $\exists X \leq t, \varphi$ stands for $\exists X(|X| \leq t \wedge \varphi)$ and $\forall X \leq t, \varphi$ stands for $\forall X(|X| \leq t \to \varphi)$, where X does not appear in term t.

The class Σ_0^B consists of all $\mathcal{L}_A^2$-formulas with no string quantifiers and only bounded number quantifiers. The class Σ_1^B consists of formulas of the form $\exists \vec{X} < \vec{t}\, \varphi$, where $\varphi \in \Sigma_0^B$ and the prefix of the bounded quantifiers might be empty. These classes are extended to Σ_i^B (and Π_i^B) for all $i \geq 0$, in the usual way. More generally we write $\Sigma_i^B(\mathcal{L})$ to denote the class of Σ_i^B-formulas which may have function and predicate symbols from $\mathcal{L} \cup \mathcal{L}_A^2$.

Two-sorted complexity classes contain relations $R(\vec{x}, \vec{X})$, where $\vec{x}$ are number arguments and $\vec{X}$ are string arguments. In defining complexity classes using machines or circuits, the number arguments are represented in unary notation and the string arguments are represented in binary. The string arguments are the main inputs, and the number arguments are auxiliary inputs that can be used to index the bits of strings.

In the two-sorted setting, we can define AC^0 to be the class of relations $R(\vec{x}, \vec{X})$ such that some alternating Turing machine accepts R in time $\mathcal{O}(\log n)$ with a constant number of alternations. Then the descriptive complexity characterization of AC^0 gives rise to the following theorem [3, Chapter 4].

▶ **Theorem 1.** *A relation $R(\vec{x}, \vec{X})$ is in AC^0 iff it is represented by some Σ_0^B-formula $\varphi(\vec{x}, \vec{X})$.*

Given a class of relations C, we associate a class FC of string-valued functions $F(\vec{x}, \vec{X})$ and number functions $f(\vec{x}, \vec{X})$ with C as follows. We require these functions to be p-bounded, i.e., $|F(\vec{x}, \vec{X})|$ and $f(\vec{x}, \vec{X})$ are bounded by a polynomial in $\vec{x}$ and $|\vec{X}|$. Then we define FC to consist of all p-bounded number functions whose graphs are in C and all p-bounded string functions whose bit graphs are in C.

▶ **Definition 2.** Let C be a complexity class. A relation $R_1(\vec{x}, \vec{X})$ is C-many-one-reducible to a relation $R_2(\vec{y}, \vec{Y})$ (written $R_1 \leq^{\mathsf{C}}_{\mathsf{m}} R_2$) if there are functions $\vec{f}, \vec{F}$ in FC such that

$$R_1(\vec{x}, \vec{X}) \leftrightarrow R_2(\vec{f}(\vec{x}, \vec{X}), \vec{F}(\vec{x}, \vec{X})).$$

A function $H_1(\vec{x}, \vec{X})$ is C-many-one-reducible to a function $H_2(\vec{y}, \vec{Y})$ if there are functions $G, \vec{f}, \vec{F}$ in FC such that

$$H_1(\vec{x}, \vec{X}) = G(H_2(\vec{f}(\vec{x}, \vec{X}), \vec{F}(\vec{x}, \vec{X}))).$$

Here we are mainly interested in the cases that C is either AC^0 or L (log space). We also need a generalization of AC^0 many-one reducibility called simply AC^0 reducibility in [3, Definition IX.1.1], which we will call AC^0 *oracle reducibility*. Roughly speaking a function or relation is AC^0-oracle-reducible to a collection $\mathcal{L}$ of functions and relations if it can be computed by a uniform polynomial size constant depth family of circuits which have unbounded fan-in gates computing functions and relations from $\mathcal{L}$ (i.e. 'oracle gates'), in addition to Boolean gates. Formally:

▶ **Definition 3.** A string function F is AC^0-oracle-reducible to a collection $\mathcal{L}$ of relations and functions (written $F \leq^{\mathsf{AC}^0}_{\mathsf{o}} \mathcal{L}$) if there is a sequence of string functions $F_1, \ldots, F_n = F$ such that each F_i is p-bounded and its bit graph is represented by a $\Sigma^B_0(\mathcal{L}, F_1, \ldots, F_{i-1})$-formula.

A number function f is AC^0-oracle-reducible to $\mathcal{L}$ if $f = |F|$ for some string function F which is AC^0-reducible to $\mathcal{L}$. A relation R is AC^0-oracle-reducible to $\mathcal{L}$ if its characteristic function is AC^0-oracle-reducible to $\mathcal{L}$.

We note that standard small complexity classes including AC^0, TC^0, NC^1, NL and P (as well as their corresponding function classes) are closed under AC^0 oracle reductions.

2.2 Two-sorted theories

The theory V^0 for AC^0 is the basis for developing theories for small complexity classes within P in [3]. V^0 has the vocabulary $\mathcal{L}^2_A$ and is axiomatized by the set of *2-BASIC* axioms, which express basic properties of symbols in $\mathcal{L}^2_A$, together with the *comprehension* axiom schema

$$\Sigma^B_0\text{-}COMP: \qquad \exists X \leq y \, \forall z < y \big(X(z) \leftrightarrow \varphi(z)\big),$$

where $\varphi \in \Sigma^B_0(\mathcal{L}^2_A)$ and X does not occur free in φ. Although V^0 has no explicit induction axiom, nevertheless, using Σ^B_0-*COMP* and the fact that $|X|$ produces the maximum element of the finite set X, the following schemes are provable in V^0 for every formula $\varphi \in \Sigma^B_0(\mathcal{L}^2_A)$

$$\Sigma^B_0\text{-}IND: \qquad \big[\varphi(0) \wedge \forall x\big(\varphi(x) \rightarrow \varphi(x+1)\big)\big] \rightarrow \forall x \varphi(x),$$
$$\Sigma^B_0\text{-}MIN: \qquad \varphi(y) \rightarrow \exists x\big(\varphi(x) \wedge \neg\exists z < x \, \varphi(z)\big).$$

In general, we say that a string function $F(\vec{x}, \vec{X})$ is Σ^B_1-definable (or provably total) in a two-sorted theory $\mathcal{T}$ if there is a Σ^B_1 formula $\varphi(\vec{x}, \vec{X}, Y)$ representing the graph $Y = F(\vec{x}, \vec{X})$ of F such that $\mathcal{T} \vdash \forall \vec{x} \forall \vec{X} \exists! Y \varphi(\vec{x}, \vec{X}, Y)$. Similarly for a number function $f(\vec{x}, \vec{X})$.

It was shown in [3, Chapter 5] that V^0 is finitely axiomatizable, and a function is provably total in V^0 if and only if it is in FAC^0.

In [3, Chapter 9], Cook and Nguyen develop a general method for associating a theory VC with certain complexity classes $\mathsf{C} \subseteq \mathsf{P}$, where VC extends V^0 with an additional axiom asserting the existence of a solution to a complete problem for C. In order for this method to work, the class C must be closed under AC^0-oracle-reducibility (Definition 3). The method shows how to define a universal conservative extension $\overline{\mathsf{VC}}$ of VC, where $\overline{\mathsf{VC}}$ has string function symbols for precisely the string functions of FC, and terms for precisely the number functions of FC. Further, $\overline{\mathsf{VC}}$ proves the $\Sigma^B_0(\mathcal{L})$-*IND* and $\Sigma^B_0(\mathcal{L})$-*MIN* schemes, where $\mathcal{L}$ is the vocabulary of $\overline{VC}$. It follows from the Herbrand Theorem that the provably total functions of both VC and $\overline{\mathsf{VC}}$ are precisely those in FC.

Using this framework Cook and Nguyen define specific theories for several complexity classes and give examples of theorems formalizable in each theory. The theories of interest to us in this paper are VTC^0, VNC^1, VNL and VP for the complexity classes TC^0, NC^1, NL and P respectively. All of these theories have vocabulary $\mathcal{L}^2_A$. Let $\langle x, y\rangle$ denote the *pairing function*, which is the $\mathcal{L}^2_A$ term $(x+y)(x+y+1)+2y$. The theory VTC^0 is axiomatized by the axioms of V^0 and the axiom:

$$NUMONES: \qquad \exists Z \leq 1 + \langle n, n\rangle, \delta_{\mathsf{NUM}}(n, X, Z), \tag{2.1}$$

where the formula $\delta_{\mathsf{NUM}}(n, X, Z)$ asserts that Z is a matrix consisting of n rows such that for every $y \leq n$, the y^{th} row of Z encodes the number of ones in the prefix of length y of the binary string X. Thus, the n^{th} row of Z essentially "counts" the number of ones in X. Because of this counting ability, VTC^0 can prove *the pigeonhole principle* $PHP(n, F)$ saying that if F maps a set of $n+1$ elements to a set of n elements, then F is not an injection.

The theory VNC^1 is axiomatized by the axioms of V^0 and the axiom:

$$MFV: \qquad \exists Y \leq 2n+1, \delta_{\mathsf{MFV}}(n, F, I, Y), \tag{2.2}$$

where F and I encode a monotone Boolean formula with n literals and its input respectively, and the formula $\delta_{\mathsf{MFV}}(n, G, I, Y)$ holds iff Y correctly encodes the evaluation of the formula encoded in F on input I. Recall that the *monotone Boolean formula value* problem is complete for NC^1.

The theory VP is axiomatized by the axioms of V^0 and the axiom *MCV*, which is defined very similarly to *MFV*, but the *monotone circuit value* problem is used instead.

The theory VNL is axiomatized by the axioms of V^0 and the axiom:

$$CONN: \qquad \exists U \leq \langle n, n\rangle + 1, \delta_{\mathsf{CONN}}(n, E, U), \tag{2.3}$$

where E encodes the edge relation of a directed graph G with n vertices $v_0, \ldots, v_{n-1}$, and the formula $\delta_{\mathsf{CONN}}(n, E, U)$ holds iff U is a matrix of n rows, where the d^{th} row encodes the set of all vertices in G that are reachable from v_0 using a path of length at most d.

Similar to what is currently known about complexity classes, it was shown in [3, Chapter 9] that $\mathsf{V}^0 \subsetneq \mathsf{VTC}^0 \subseteq \mathsf{VNC}^1 \subseteq \mathsf{VNL} \subseteq \mathsf{VP}$.

2.3 The Ccv problem and its complexity classes

A *comparator gate* is a function $C : \{0,1\}^2 \to \{0,1\}^2$, that takes an input pair (p, q) and outputs a pair $(p \wedge q, p \vee q)$. Intuitively, the first output in the pair is the smaller bit among the two input bits p, q, and the second output is the larger bit.

We will use the graphical notation in Fig. 2 to denote a comparator gate, where x and y denote the names of the wires, and the direction of the arrow denotes the direction to which we move the larger bit as shown in the picture.

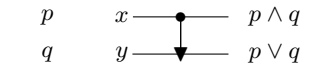


Figure 2

A *comparator circuit* is a directed acyclic graph consisting of: *input nodes* with in-degree zero and out-degree one, *output nodes* with in-degree one and out-degree zero, and *internal nodes* with in-degree two and out-degree two, where each internal node is labelled with a comparator gate. We also require each output computed by a comparator gate has fan-out exactly one. Under this definition, each comparator circuit can be seen as consisting of the wires that carry the bit values and are arranged in parallel, and each comparator gate connects exactly two wires as previously shown in Fig. 1.

The *comparator circuit value* problem (Ccv) is the task of deciding, on a given input assignment, if a designated wire of a comparator circuit outputs one.

▶ **Definition 4.** The complexity class CC (resp. $\mathsf{CC}^{\mathsf{Subr}}$, CC^{*}) is the class of decision problems (i.e. relations) that are AC^0 many-one-reducible (resp. log space-reducible, AC^0 oracle-reducible) to Ccv. A decision problem R is CC-complete (resp. $\mathsf{CC}^{\mathsf{Subr}}$-complete, CC^{*}-complete) if the respective class is the closure of R under the corresponding reducibility. We say that R is $\mathsf{CC}_{\mathsf{all}}$-complete if it is complete in all three senses.

The next result is a straightforward consequence of (1.1) and the definitions involved.

▶ **Lemma 5.** *If a decision problem is* CC*-complete then is is* $\mathsf{CC}_{\mathsf{all}}$*-complete.*

In the above definition of comparator circuit, each comparator gate can point in either direction, upward or downward (see Fig. 1). However, it is not hard to show the following.

▶ **Proposition 1.** *The* Ccv *problem with the restriction that all comparator gates point in the same direction is* CC*-complete.*

2.4 The stable marriage problem

An instance of the stable marriage problem (Sm) is given by a number n (specifying the number of men and the number of women), together with a preference list for each man and each woman specifying a total ordering on all people of the opposite sex. The goal of Sm is to produce a perfect matching between men and women, i.e., a bijection from the set of men to the set of women, such that the following *stability* condition is satisfied: there are no two people of the opposite sex who like each other more than their current partners. Such a stable solution always exists, but it may not be unique. Under this formulation Sm is a *search problem*, rather than a function problem.

However there is always a unique *man-optimal* and a unique *woman-optimal* solution. In the man-optimal solution each man is matched with a woman whom he likes at least as well as any woman that he is matched with in any stable solution. Dually for the woman-optimal solution. Thus both the man-optimal and the woman-optimal versions are function problems (and hence equivalent to decision problems.)

We show here that the search version and the decision versions are computationally equivalent, and each is complete for CC. Section 6.1 shows how to reduce the lexicographical first maximal matching problem (a decision problem complete for CC) to the search version of Sm, and Section 6.2 shows how to reduce both the man-optimal and the woman-optimal function problems of Sm to Ccv.

2.5 Notation

We write the notation "$(T \vdash)$" in front of the statement of a theorem to indicate that the statement is formulated and proved within the theory T.

3 The new theory VCC^*

We encode a comparator circuit as a sequence of pairs $\langle i, j\rangle$, where each pair $\langle i, j\rangle$ encodes a comparator gate that swaps the values of the wires i and j if and only if the value on wire i is greater than the value of wire j. We also allow "dummy" gates of the form $\langle i, i\rangle$, which do nothing. We want to define a formula $\delta_{\mathsf{CCV}}(m, n, X, Y, Z)$, where

- X encodes a comparator circuit with m wires and n gates as a sequence of n pairs $\langle i, j\rangle$ with $i, j < m$, and we write $(X)^i$ to denote the i^{th} comparator gate of the circuit.
- $Y(i)$ encodes the input value for the i^{th} wire as a truth value, and
- Z is an $(n+1) \times m$ matrix, where $Z(i, j)$ is the value of wire j at layer i, where each layer is simply a sequence of the values carried by all the wires right after a comparator gate.

Although X encodes a circuit with only n gates, Z actually encodes $n+1$ layers since we use the first layer to encode the input values of the circuit. The formula $\delta_{\mathsf{CCV}}(m, n, X, Y, Z)$ holds iff Z encodes the correct values of the layers computed by the comparator circuit encoded by X with input Y, and thus is defined as the following Σ^B_0-formula:

$$\begin{aligned} &\forall i < m\big(Y(i) \leftrightarrow Z(0, i)\big) \wedge \forall i < n \forall x < m \forall y < m, \\ &\qquad (X)^i = \langle x, y\rangle \rightarrow \left[\begin{array}{ll} & Z(i+1, x) \leftrightarrow \big(Z(i, x) \wedge Z(i, y)\big) \\ \wedge & Z(i+1, y) \leftrightarrow \big(Z(i, x) \vee Z(i, y)\big) \\ \wedge & \forall j < m\Big[(j \neq x \wedge j \neq y) \rightarrow \big(Z(i+1, j) \leftrightarrow Z(i, j)\big)\Big] \end{array}\right] \end{aligned} \tag{3.1}$$

Note that in this paper we index the entries of matrices starting from 0 instead of 1.

▶ **Definition 6.** The theory VCC^* has vocabulary $\mathcal{L}^2_A$ and is axiomatized by the axioms of V^0 and the following axiom (the formula $\delta_{\mathsf{CCV}}(m, n, X, Y, Z)$ is defined as in (3.1)):

$$CCV: \qquad \exists Z \leq \langle m, n+1\rangle + 1,\ \delta_{\mathsf{CCV}}(m, n, X, Y, Z) \tag{3.2}$$

There is a technical lemma required to show that VCC^* fits the framework described in [3, Chapter 9]. Define $F_{\mathsf{CCV}}(m, n, X, Y)$ to be the Z satisfying $\delta_{\mathsf{CCV}}(m, n.X, Y, Z)$ (with each $Z(i)$ set false when it is not determined). We need to show that the *aggregate* F^*_{CCV} of F_{CCV} is Σ^B_1-definable in VCC^*, where (roughly speaking) F^*_{CCV} is the string function that gathers the values of F_{CCV} for a polynomially long sequence of arguments. The nature of CC circuits makes this easy: The sequence of outputs for a sequence of circuits can be obtained from a single circuit which computes them all in parallel: the lines of the composite circuit comprise the union of the lines of each component circuit. Thus the framework of [3, Chapter 9] does apply to VCC^*, and in particular the theory $\overline{\mathsf{VCC}^*}$ is a universal conservative extension of VCC^* whose function symbols are precisely those in the function class FCC.

It is hard to work with VCC^* up to this point since we have not shown whether VCC^* can prove the definability of basic counting functions (as in VTC^0). However, we have the following theorem.

▶ **Theorem 7** ($\mathsf{VNC}^1 \subseteq \mathsf{VCC}^*$). *The theory VCC^* proves the axiom MFV defined in (2.2).*

Proof. Observe that each comparator gate can produce simultaneously an AND gate and an OR gate with the only restriction that each of these gates must have fan-out one. However,

since all AND and OR gates of a monotone Boolean formula also have fan-out one, each instance of the Boolean formula value problem is a special case of Ccv. ◀

A corollary of this theorem is that $\mathsf{VTC}^0 \subseteq \mathsf{VCC}^*$, and thus we can use the counting ability of VTC^0 freely in VCC^* proofs.

▶ **Theorem 8** ($\mathsf{VCC}^* \subseteq \mathsf{VP}$). *The theory* VP *proves the axiom CCV defined in (3.2).*

Proof. This follows since Ccv is a special case of the *monotone circuit value* problem. ◀

4 Lexicographical first maximal matching problem is CC-complete

Let $G = (V, W, E)$ be a bipartite graph, where $V = \{v_i\}_{i=0}^{m-1}$, $W = \{w_i\}_{i=0}^{n-1}$ and $E \subseteq V \times W$. The *lexicographical first maximal matching* (lfm-matching) is the matching produced by successively matching each vertex $v_0, \ldots, v_{m-1}$ to the least vertex available in W. The lexicographical first maximal matching problem (Lfmm) is to decide if a designated edge belongs to the lfm-matching of G, and 3Lfmm is the restriction of Lfmm to graphs of degree at most three. In this section we give simplified constructions showing that Ccv is AC^0-many-one-reducible to 3Lfmm and Lfmm is AC^0-many-one-reducible to Ccv.

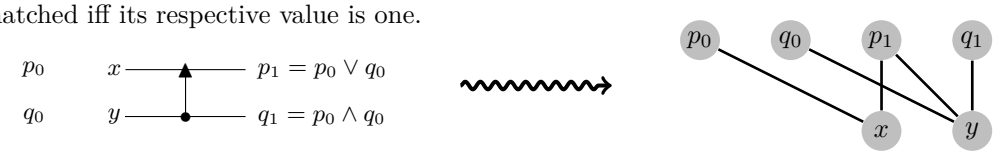
Formally, let $E_{m\times n}$ be a matrix encoding the edge relation of a bipartite graph with m bottom nodes and n top nodes, where $E(i,j) = 1$ iff the bottom node v_i is adjacent to the top node w_j. Let L be a matrix of the same size as E with the following intended interpretation: $L(i,j) = 1$ iff the edge (v_i, w_j) is in the lfm-matching. We can define a Σ_0^B-formula $\delta_{\mathsf{LFMM}}(m,n,X,L)$, which holds iff L properly encodes the lfm-matching of the bipartite graph represented by X as follows:

$$\forall i < m \forall j < n,\ L(i,j) \leftrightarrow \left[\begin{array}{lll} E(i,j) & \wedge & \forall k < j\, \forall \ell < i \big(\neg L(i,k) \wedge \neg L(\ell,j)\big) \\ & \wedge & \forall k < j \big(\neg E(i,k) \vee \exists i' < i\, L(i',k)\big) \end{array}\right]. \tag{4.1}$$

4.1 Ccv $\leq_{\mathsf{m}}^{\mathsf{AC}^0}$ 3Lfmm

By Proposition 1, it suffices to consider only instance of Ccv, where all comparator gates point upward. We will show that these instances of Ccv are AC^0-many-one-reducible to instances of 3Lfmm, which consist of bipartite graphs with *degree at most three*.

The key observation is that a comparator gate on the left below closely relates to an instance of 3Lfmm on the right. We use the top nodes p_0 and q_0 to represent the values p_0 and q_0 carried by the wires x and y respectively before the comparator gate, and the nodes p_1 and q_1 to represent the values of x and y after the comparator gate, where a top node is matched iff its respective value is one.



If nodes p_0 and q_0 are not previously matched, i.e. $p_0 = q_0 = 0$ in the comparator circuit, then edges $\langle x, p_0\rangle$ and $\langle y, q_0\rangle$ are added to the lfm-matching. So the nodes p_1 and q_1 are not matched. If p_0 is previously matched, but q_0 is not, then edges $\langle x, p_1\rangle$ and $\langle y, q_0\rangle$ are added to the lfm-matching. So the node p_1 will be matched but q_1 will remain unmatched. The other two cases are similar.

Thus, we can reduce a comparator circuit to the bipartite graph of an 3Lfmm instance by converting each comparator gate into a "gadget" described above. We will describe our method through an example, where we are given the comparator circuit in Fig. 3.

We divide the comparator circuit into vertical layers 0, 1, and 2 as shown in Fig. 3. Since the circuit has three wires a, b and c, for each layer i, we use six nodes, including three top nodes a_i, b_i and c_i representing the values of the wires a, b and c respectively, and three bottom nodes a'_i, b'_i, c'_i, which are auxiliary nodes used to simulate the effect of the comparator gate at layer i.

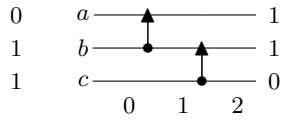
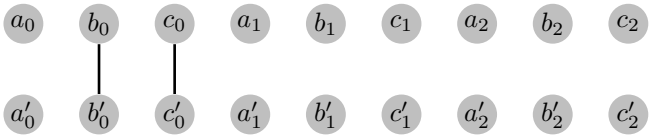
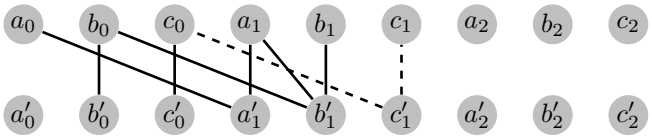


Figure 3

Layer 0: This is the input layer, so we add an edge $\{x_i, x'_i\}$ iff the wire x takes input 1. In this example, since b and c are wires taking input 1, we add the edges $\{b_0, b'_0\}$ and $\{c_0, c'_0\}$.

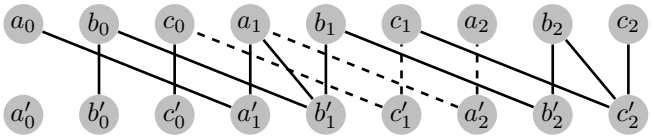


Layer 1: We then add the gadget simulating the comparator gate from wire b to wire a.



Since the value of wire c does not change when going from layer 0 to layer 1, we can simply propagate the value of c_0 to c_1 using the pair of dotted edges in the picture.

Layer 2: We proceed very similarly to layer 1 to get the following bipartite graph.



Finally, we can get the output values of the comparator circuit by looking at the "output" nodes a_2, b_2, c_2 of this bipartite graph. We can easily check that a_2 is the only node that remains unmatched, which corresponds exactly to the only zero produced by wire a of the comparator circuit above.

The construction above is an AC^0 many-one reduction since each gate in the comparator circuit can be reduced to exactly one gadget in the bipartite graph that simulates the effect of the comparator gate. Note that since it can be tedious and unintuitive to work with AC^0-circuits, it might seem hard to justify that our reduction is an AC^0-function. However, thanks to Theorem 1, we do not have to work with AC^0-circuits directly; instead, it is not hard to construct a Σ_0^B-formula that defines the above reduction. The correctness of our construction can be proved in VCC^* by using Σ_0^B induction on the layers of the circuits and arguing that the matching information of the nodes in the bipartite graph can be correctly translated to the values carried by the wires at each layer.

4.2 LFMM $\leq_{\mathsf{m}}^{\mathsf{AC}^0}$ CCV

Consider an instance of LFMM consisting of a bipartite graph on the left of Fig. 4. Recall that we find the lfm-matching by matching the bottom nodes $x, y, \ldots$ successively to the first available node on the top. Hence we can simulate the matching of the bottom nodes to the top nodes using comparator circuit on the right of Fig. 4, where we can think of the moving of a one, say from wire x to wire a, as the matching of node x to node a in the original bipartite graph. Note that we draw bullets without any arrows going out from them in the circuit to denote dummy gates, which do nothing. These dummy gates are introduced for

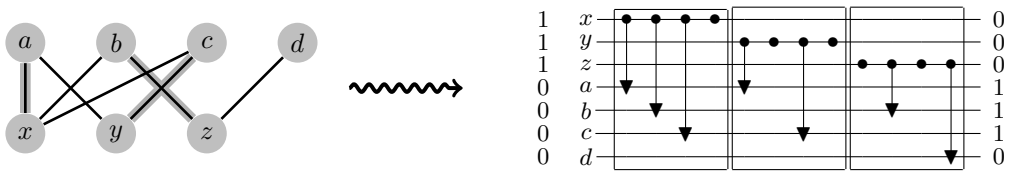


Figure 4

the following technical reason. Since the bottom nodes might not have the same degree, the position of a comparator gate really depends on the number of edges that do not appear in the bipartite graph, which makes it harder to give a direct AC^0-reduction. By using dummy gates, we can treat the graph as if it is a complete bipartite graph, where the missing edges are represented by dummy gates. This can easily be shown to be an AC^0-reduction from Lfmm to Ccv, and its correctness can be carried out in VCC^*. This together with the reduction from Section 4.1 gives us the following theorem.

▶ **Theorem 9.** ($\mathsf{VCC}^* \vdash$) *The* Lfmm *problem is* CC*-complete.*

Since the reduction from Ccv to Lfmm in Section 4.1 only produces bipartite graphs with degree at most three, we have the following corollary.

▶ **Corollary 10.** *(*$\mathsf{VCC}^* \vdash$*) The* 3Lfmm *problem is* CC*-complete.*

5 The theory VCC^* contains VNL

Each instance of the Reachability problem consists of a directed acyclic graph $G = (V, E)$, where $V = \{v_0, \ldots, v_{n-1}\}$, and we want to decide if there is a path from v_0 to v_{n-1}. It is well-known that Reachability is NL-complete. It is also well-known that the Reachability problem still remains NL-complete under the following restriction:

$$\text{The graph } G \text{ only has directed edges of the from } (v_i, v_j), \text{ where } i < j. \tag{5.1}$$

We will show how to use comparator circuits to solve the above restricted instances of Reachability. We believe that our new construction is more intuitive than the one in [8, 6]. Moreover, we reduce Reachability to Ccv directly without going through some intermediate complete problem, and this was stated as an open problem in [8, Chapter 7.8.1].

We will demonstrate our construction through a simple example, where we have the directed graph in Fig. 5 satisfying the assumption (5.1). We will build a comparator circuit as in Fig. 6, where the wires $\nu_0, \ldots, \nu_4$ represent the vertices $v_0, \ldots, v_4$ of the preceding graph and the wires $\iota_0, \ldots, \iota_4$ are used to feed 1-bits into the wire v_0, and from there to the other wires v_i reachable from v_0. We let every wire ι_i take input one and every wire ν_i take input zero. We next show how to construct the gadget in the boxes. For a graph with n vertices ($n = 5$ in our example), the gadget in the ℓ^{th} box is constructed as follows:

```
1: Add a comparator gate from wire ι_ℓ to wire ν_0
2: for i = 0, ..., n − 1 do
3:   for j = i + 1, ..., n − 1 do
4:     Add a comparator gate from ν_i to ν_j if (v_i, v_j) ∈ E,
       or a dummy gate on ν_i otherwise.
5:   end for
6: end for
```

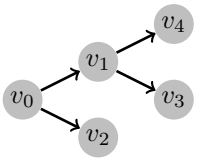


Figure 5

Note that we only use the loop structure to clarify the order the gates are added. The construction can easily be done in AC^0 since the position of each gate can be calculated exactly, and thus all gates can be added independently from one another. Note that for a

graph with n vertices, we have at most n vertices reachable from a single vertex, and thus we need n gadgets described above. In our example, there are at most 5 wires reachable from wire ν_0, and thus we utilize the gadget 5 times.

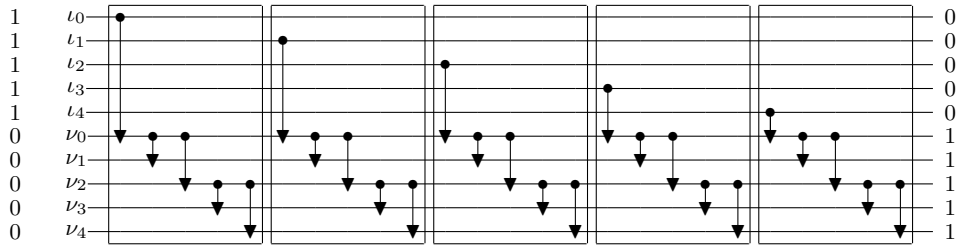


Figure 6 A comparator circuit that solves REACHABILITY. (The dummy gates are omitted.)

Intuitively, the construction works since each gadget from a box looks for the *lexicographical first maximal path* starting from v_0 (with respect to the *natural lexicographical ordering* induced by the vertex ordering $v_0, \ldots, v_n$), and then the vertex at the end of the path will be marked (i.e. its wire will now carry one) and thus excluded from the search of the gadgets that follow. For example, the gadget from the left-most dashed box in Fig. 6 will move a one from wire ι_0 to wire ν_0 and from wire ν_0 to wire ν_1. This essentially "marks" the wire ν_1 since we cannot move the one away from ν_1, and thus ν_1 can no longer receive any new incoming ones. Hence, the gadget from the second box in Fig. 6 will repeat the process of finding the lex-first maximal path from v_0 to the remaining (unmarked) vertices. These searches end when all vertices reachable from v_0 are marked. Note that this has the same effect as applying the *depth-first search* algorithm to find all the vertices reachable from v_0. Thus, we can prove the following theorem.

▶ **Theorem 11** ($\mathsf{VNL} \subseteq \mathsf{VCC}^*$). *The theory* VCC^* *proves the axiom CONN defined in (2.3).*

As a of consequence of Theorem 11, we have the following result.

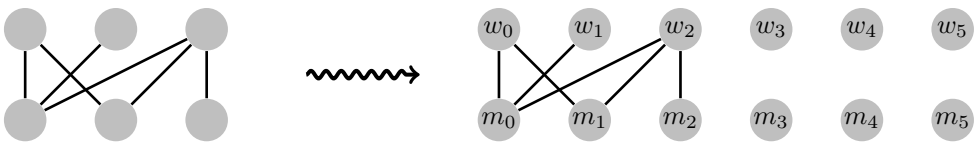
▶ **Theorem 12.** CC^* *is closed under many-one* NL*-reductions, and hence* $\mathsf{CC}^{\mathsf{Subr}} \subseteq \mathsf{CC}^*$.

Proof. This follows from the following three facts: The function class FCC^* is closed under composition, $\mathsf{FNL} \subseteq \mathsf{FCC}$, and a decision problem is in CC^* if and only if its characteristic function is in FCC^*. ◀

6 The SM problem is CC-complete

6.1 3LFMM $\leq_{\mathsf{m}}^{\mathsf{AC}^0}$ SM

Let $G = (V, W, E)$ be a bipartite graph from an instance of 3LFMM, where V is the set of bottom nodes, W is the set of top nodes, and E is the edge relation such that the degree of each node is at most three (see the example in the figure on the left below). Without loss of generality, we can assume that $|V| = |W| = n$. To reduce it to an instance of SM, we double the number of nodes in each partition, where the new nodes are enumerated after the original nodes and the original nodes are enumerated using the ordering of the original bipartite graph, as shown in the diagram on the right below. We also let the bottom nodes and top nodes represent the men and women respectively.



It remains to define a preference list for each person in this SM instance. The preference list of each man m_i, who represents a bottom node in the original graph, starts with all the woman w_j (at most three of them) adjacent to m_i in the order that these women are enumerated, followed by all the women $w_n, \ldots, w_{2n-1}$; the list ends with all women w_j not adjacent to m_i also in the order that they are enumerated. For example, the preference list of m_2 in our example is $w_2, w_3, w_4, w_5, w_0, w_1$. The preference list of each newly introduced man m_{n+i} simply consists of $w_0, \ldots, w_{n-1}, w_n, \ldots, w_{2n-1}$, i.e., in the order that the top nodes are listed. Preference lists for the women are defined dually.

Intuitively, the preference lists are constructed so that any stable marriage (not necessarily man-optimal) of the new SM instance must contain the lfm-matching of G. Furthermore, if a bottom node u from the original graph is not matched to any top node in the lfm-matching of G, then the man m_i representing u will marry some top node w_{n+j}, which is a dummy node that does not correspond to any node of G. Thus we have the following theorem.

▶ **Theorem 13.** *($\mathsf{VCC}^* \vdash$) The* 3LFMM *problem is* AC^0*-many-one-reducible to* SM*.*

6.2 $\text{SM} \leq_{\mathsf{m}}^{\mathsf{AC}^0}$ CCV

In this section, we formalize a reduction from SM to CCV due to Subramanian [8, 9]. Subramanian did not reduce SM to CCV directly, but to the *network stability problem* built from the less standard X gate, which takes two inputs p and q and produces two outputs $p' = p \wedge \neg q$ and $q' = \neg p \wedge q$. It is important to note that the "*network*" notion in Subramanian's work denotes the generalization of circuits by allowing connection from output of a gate to input of any gate including itself, and thus a network in his definition might contain cycles. An X-network is a network consisting only of X gates under the important restriction that each X gate has fan-out exactly one for each output it computes. The network stability problem for X gate (XNS) is then to decide if an X-network has a stable configuration, i.e., a way to assign Boolean values to the wires of the network so that the values are compatible with all the X gates of the network. Subramanian showed in his dissertation [8] that SM, XNS and CCV are all equivalent under log space reduction.

We do not work with XNS in this paper since networks are less intuitive and do not have a nice graphical representation as do comparator circuits. By utilizing Subramanian's idea, we give a more direct AC^0-reduction from SM to CCV. For this goal, it turns out to be conceptually simpler to go through a new variant of CCV, where the comparator gates are three-valued instead of Boolean.

6.2.1 THREE-VALUED CCV is CC-complete

We define the THREE-VALUED CCV problem similarly to CCV, except each wire can now take any of the values 0, 1 or $*$. A wire takes value $*$ when its value is not known to be 0 or 1. The two outputs of a three-valued comparator gate on inputs p and q is defined as follows.

$$p \wedge q = \begin{cases} 0 & \text{if } p = 0 \text{ or } q = 0 \\ 1 & \text{if } p = q = 1 \\ * & \text{otherwise.} \end{cases} \qquad p \vee q = \begin{cases} 0 & \text{if } p = q = 0 \\ 1 & \text{if } p = 1 \text{ or } q = 1 \\ * & \text{otherwise.} \end{cases}$$

Every instance of CCV is also an instance of THREE-VALUED CCV. It is also not hard to show that every instance of THREE-VALUED CCV is AC^0-reducible to an instance of CCV by using a pair of Boolean wires to represent each three-valued wire and adding comparator gates appropriately to simulate three-valued comparator gates. Thus, the THREE-VALUED CCV problem is CC-complete.

6.2.2 A fixed-point method for SM

We formalize a reduction from SM to THREE-VALUED CCV based on [8, 9]. Consider an instance of SM, where preference matrices for men a, b and women x, y are given in Fig. 7. From this instance of SM, we construct a three-valued comparator circuit in Fig. 7 as follows.

Men:

a	x	y
b	y	x

Women:

x	a	b
y	a	b

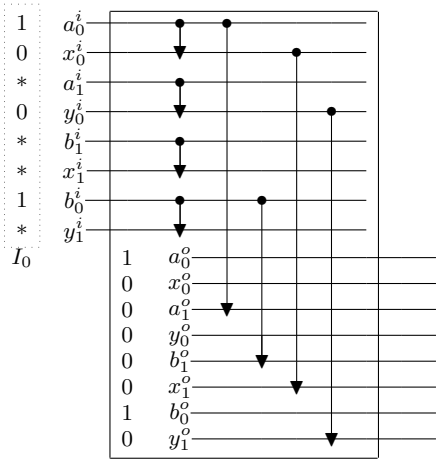


Figure 7

First, since we have two men a, b and two women x, y, we start with four pairs of wires (a_0^i, x_0^i), (a_1^i, y_0^i), (b_0^i, y_1^i), and (b_1^i, x_1^i), connected by four gates $\langle a_0^i, x_0^i\rangle$, $\langle a_1^i, y_0^i\rangle$, $\langle b_0^i, y_1^i\rangle$ and $\langle b_1^i, x_1^i\rangle$ respectively, which represent four possible ways of pairing men a, b to women x, y. The subscripts are important in our construction since the subscript of a person p within a pair indicates the preference of a person about his or her partner in the pair; the superscripts i are less important, and used to indicate that all of these wires are the 'input wires' of this construction. For example, the subscript of b in the pair (b_0^i, y_1^i) indicates that y is a's first choice, and the subscript of y in this pair indicates that b is y's second choice. For convenience, let Pair be a binary predicate such that $\mathsf{Pair}(m_j^i, w_k^i)$ holds iff m is a man and w is a woman and wires m_j^i and w_k^i are paired up, i.e., w is at the j^{th} position of m's preference list and m is at the k^{th} position of w's preference list.

Second, we will introduce four more pairs of 'output wires' (a_0^o, x_0^o), (a_1^o, y_0^o), (b_0^o, y_1^o), and (b_1^o, x_1^o), which are arranged in exactly the same order as input wires, where the subscripts follow the same preference rules as with the input wires. We also define $\mathsf{Pair}(m_j^o, w_k^o)$ to hold iff m is man and w is woman and m_j^o and w_k^o are paired up. Since all subscripts of the wires encode the preference information, they can be used in our construction as follows. Assume that preference lists are of size n, then for every person p, we add a gate from wire p_j^i to p_{j+1}^o for every $j < n-1$. In our example, we add four gates $\langle a_0^i, a_1^o\rangle$, $\langle b_0^i, b_1^o\rangle$, $\langle x_0^i, x_1^o\rangle$, and $\langle y_0^i, y_1^o\rangle$ as shown in Fig. 7. Note that these gates can be added in any order. It remains to show how to feed inputs to the 'output wires'. We let output wire m_0^o take input one for every man m, and let the rest of output wires have zero inputs.

Given an instance of SM with n men and n women, define $\mathcal{M} : \{0,1,*\}^{2n^2} \to \{0,1,*\}^{2n^2}$ to be the function computed by the preceding circuit construction, where the inputs of $\mathcal{M}$ are those fed into the input wires, and the outputs of $\mathcal{M}$ are those produced by the output wires. We will use the following notation. Any sequence $I \in \{0,1,*\}^{2n^2}$ can be seen as an input of function $\mathcal{M}$, and thus we write $I(p_j^i)$ to denote the input value of wire p_j^i with respect to I. Similarly, if a sequence $J \in \{0,1,*\}^{2n^2}$ is an output of $\mathcal{M}$, then we write $J(p_j^o)$

to denote the output value of wire p_j^o.

Let sequence $I_0 \in \{0,1,*\}^{2n^2}$ be an input of $\mathcal{M}$ defined as follows: $I_0(m_0^i) = 1$ for every man m, and $I_0(w_0^i) = 0$ for every woman w, and $I_0(p_j^i) = *$ for every person p and every j, $1 \leq j < n$. Note that the number of $*$'s in the sequence I_0 is

$$c(n) = 2n^2 - 2n. \tag{6.1}$$

Our version of Subramanian's method [8, 9] consists of computing

$$I_{c(n)} = \mathcal{M}^{c(n)}(I_0),$$

where $\mathcal{M}^d$ simply denotes the d^{th} power of $\mathcal{M}$, i.e. the function we get by composing $\mathcal{M}$ with itself d times. It turns out that $I_{c(n)}$ is a fixed point of $\mathcal{M}$, i.e. $I_{c(n)} = \mathcal{M}(I_{c(n)})$. To show this, we define a sequence I' to be an *extension* of a sequence I if $I(p) = I'(p)$ for every person p such that $I(p) \in \{0,1\}$. We can show that $\mathcal{M}(I)$ is an extension of I for every I which extends I_0, and hence $\mathcal{M}^d(I_0)$ extends I_0 for all d. It follows that $\mathcal{M}^{c(n)}(I_0)$ is a fixed point because there are at most $c(n)$ $*$'s to convert to 0 or 1.

Now we can extract a stable marriage from the fixed point $I_{c(n)}$ by letting B be the sequence obtained by substituting zeros for all remaining $*$-values in $I_{c(n)}$. Then B is also a fixed point of $\mathcal{M}$. A stable marriage can then be extracted from B by announcing the marriage of a man m and a woman w if and only if $\mathsf{Pair}(m_j^o, w_k^o)$ and $B(m_j^o) = 1$ and $B(w_k^o) = 0$. Our goal is to formalize the correctness of this method.

In the example in Fig. 7, we can check that the fixed point $I_4 = \mathcal{M}^4(I_0)$ in this case simply consists of Boolean values, where $(I_4(a_0^o), I_4(x_0^o)) = (1,0)$ and $(I_4(b_0^o), I_4(y_1^o)) = (1,0)$. Thus, women x, y are married to men a, b respectively, which is a stable marriage.

More formally, given a three-valued sequence I, let $I[* \to v]$ denote the sequence we get by substituting v for all the $*$-values in I. Define G to be an AC^0-function, which takes as input a Boolean fixed point B of $\mathcal{M}$, and returns a marriage M in the way explained above. (Note that since $B = \mathcal{M}(B)$, we have $B(p_k^i) = B(p_k^o)$ for every person p and every $k < n$; however, the superscripts o and i are useful for distinguishing between input and output values of the comparator circuit computing $\mathcal{M}$.) We can prove the following theorem.

▶ **Theorem 14.** *($\forall\mathsf{CC}^* \vdash$) Let M be a stable marriage of the* Sm *instance $\mathcal{I}$. We let $M_0 = G(I_{c(n)}[* \to 0])$ and $M_1 = G(I_{c(n)}[* \to 1])$. Then M_0 and M_1 are stable marriages, and every man gets a partner in M_0 no worse than the one he gets in M, and every woman gets a partner in M_1 no worse than the one she gets in M. In other words, M_0 and M_1 are the man-optimal and woman-optimal solutions respectively.*

Corollary 10 and Theorems 13 and 14 give us the following corollary.

▶ **Corollary 15.** *($\forall\mathsf{CC}^* \vdash$) The* Sm *problem is* CC*-complete.*

Proof. Following the above construction, we can write a Σ_0^B-formula defining an AC^0 function that takes as input an instance of Sm with preference lists for all the men and women, and produces a three-valued comparator circuit that computes the three-valued fixed point $I_{c(n)} = \mathcal{M}^{c(n)}(I_0)$, and then extracts the man-optimal stable marriage from $I_{c(n)}[* \to 0]$. Thus the man-optimal (and similarly the woman-optimal) decision versions of Sm are AC^0-many-one-reducible to Three-valued Ccv, and hence also to Ccv. Corollary 10 shows that 3Lfmm is CC-complete, and Theorem 13 shows that 3Lfmm is AC^0-many-one-reducible to Sm. Hence, Sm is CC-complete under AC^0 many-one reductions. ◀

7 Conclusion and future work

Our correctness proof of the reduction from SM to CCV is a nice example showing the utility of three-valued logic for reasoning about uncertainty. Since an instance of SM might not have a unique solution, the fact that the fixed point $I_{c(n)} = \mathcal{M}^{c(n)}(I_0)$ is three-valued indicates that the construction cannot fully determine how all the men and women can be matched. Thus, different Boolean fixed-point extensions of $I_{c(n)}$ give us different stable marriages.

It is worth noting that Subramanian's method is not the "textbook" method for solving SM. The most well-known is the Gale-Shapley algorithm [4]. In fact, our original motivation was to formalize the correctness of the Gale-Shapley algorithm, but we do not know how to talk about the computation of the Gale-Shapley algorithm in VCC^*. Thus, we leave open the question whether VCC^* proves the correctness of the Gale-Shapley algorithm.

We believe that CC deserves more attention, since on the one hand it contains interesting complete problems, but on the other hand we have no real evidence (for example based on relativized inclusions) concerning whether CCV is complete for P, and if not, whether it is comparable to NC. The perfect matching problem (for bipartite graphs or general undirected graphs) shares these same open questions with CCV. However several randomized NC^2 algorithms are known for perfect matching [5, 7], but no randomized NC algorithm is known for any CC-complete problem.

Another open question is whether the three CCV complexity classes mentioned in (1.1) coincide, which is equivalent to asking whether CC (the closure of CCV under AC^0 many-one reductions) is closed under AC^0 oracle reductions, or equivalently whether the function class FCC is closed under composition. A possible way to show this would be to show the existence of universal comparator circuits, but we do not know whether such circuits exist.

The analogous question for standard complexity classes such as TC^0, L, NL, NC, P has an affirmative answer. That is, each class can be defined as the AC^0 many-one closure of a complete problem, and the result turns out to be also closed under AC^0 oracle reducibilities. (A possible exception is the function class $\#\mathsf{L}$, whose AC^0 oracle closure is the $\#\mathsf{L}$ hierarchy [1]. This contains the integer determinant as a complete problem.)

References

1 E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *RAIRO, Theoretical Informatics and Applications*, 30(1):1–21, 1996.

2 K.E. Batcher. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computer Conference 32*, pages 307–314. ACM, 1968.

3 S. Cook and P. Nguyen. *Logical foundations of proof complexity*. Cambridge University Press, 2010.

4 D. Gale and L.S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

5 R.M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

6 E.W. Mayr and A. Subramanian. The complexity of circuit value and network stability. *Journal of Computer and System Sciences*, 44(2):302–323, 1992.

7 K. Mulmuley, U.V. Vazirani, and V.V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

8 A. Subramanian. *The computational complexity of the circuit value and network stability problems*. PhD thesis, Dept. of Computer Science, Stanford University, 1990.

9 A. Subramanian. A new approach to stable matching problems. *SIAM Journal on Computing*, 23(4):671–700, 1994.

Relating Two Semantics of Locally Scoped Names

Steffen Lösch and Andrew M. Pitts

University of Cambridge Computer Laboratory
Cambridge CB3 0FD, UK

Abstract

The operational semantics of programming constructs involving locally scoped names typically makes use of stateful *dynamic allocation*: a set of currently-used names forms part of the state and upon entering a scope the set is augmented by a new name bound to the scoped identifier. More abstractly, one can see this as a transformation of local scopes by expanding them outward to an implicit top-level. By contrast, in a neglected paper from 1994, Odersky gave a stateless lambda calculus with locally scoped names whose dynamics contracts scopes inward. The properties of 'Odersky-style' local names are quite different from dynamically allocated ones and it has not been clear, until now, what is the expressive power of Odersky's notion. We show that in fact it provides a direct semantics of locally scoped names from which the more familiar dynamic allocation semantics can be obtained by continuation-passing style (CPS) translation. More precisely, we show that there is a CPS translation of typed lambda calculus with dynamically allocated names (the Pitts-Stark $\boldsymbol{\nu}$-calculus) into Odersky's $\boldsymbol{\lambda\nu}$-calculus which is computationally adequate with respect to observational equivalence in the two calculi.

1998 ACM Subject Classification F.3.2 Operational semantics; F.3.3 Functional constructs; F.4.1 Lambda calculus and related systems

Keywords and phrases Local names, continuations, typed $\boldsymbol{\lambda}$-calculus, observational equivalence

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.396

1 Introduction

Locally scoped names are a ubiquitous feature of programming languages. Here we will be concerned with properties of this notion that are independent of the nature of the entities being named, be they mutable storage cells, objects, exceptions, communication channels, cryptographic keys, or whatever. The only assumption that we make about names is that the ambient programming language has the ability to test them for equality. The operational semantics of such locally scoped names is commonly specified in terms of *dynamically allocated fresh names*, also known as *generative names*. This is a state-based explanation of the meaning of the scoping construct: to execute a program with a locally scoped name, the current state is augmented with a fresh name and the body of the scope is executed with the scoped name bound to the fresh one. The combination of this simple mechanism with other features, especially higher-order functions as occurs in the ML family of languages, can result in programs with very complicated behaviour. The Pitts-Stark ν-*calculus* [15, 20] was intended to make this point, taking the measure of behaviour to be *observational equivalence* (also known as *contextual equivalence*), the relation between two programs of having the same observable behaviour when placed in any program context. Syntactically, the ν-calculus is simply-typed λ-calculus over ground types Name and Bool (for names and booleans respectively), augmented with a construct $\nu a.\, t$ for restricting the scope of a name a to a term t. The ν-calculus is given an operational semantics that makes it a fragment of Standard ML [10] by interpreting Name as ML's type `unit ref` of references to

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 396–411

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

the unit value and taking $\nu a.\, t$ to be `let a = ref() in t`. The properties of observational equivalence for the ν-calculus turn out to be remarkably complex, despite the simplicity of the language. See [2, Sect. 1] for a survey of the literature on the ν-calculus.

The ν-calculus combines dynamically allocated local names with higher-order functions. But is dynamic allocation the only way to interpret the meaning of locally scoped names? In fact there is another, but much less well known semantics for them. At about the same time that the ν-calculus was introduced, Odersky developed what he called the *$\lambda\nu$-calculus* [12]. Syntactically this is essentially identical to the ν-calculus; it has pairs as well as functions, but the ν-calculus could have had those too (we add them here). However, the local scoping construct $\nu a.\, t$ is given a very different semantics, which we recall in Sect. 4. On the one hand, its most important feature is that it is stateless, or 'referentially transparent'; and Odersky shows that $\lambda\nu$-calculus is a conservative extension of λ-calculus with respect to observational equivalence. On the other hand, it has some properties that seem very strange compared with the more familiar, generative interpretation. For instance, dynamic allocation of locally scoped names generally does not commute with function abstraction; whereas in Odersky's calculus, $\nu a.\, \lambda x \to t$ is observationally equivalent to (indeed, reduces to) $\lambda x \to \nu a.\, t$. (For example, in the ν-calculus $\nu a.\, \lambda x \to a$ and $\lambda x \to \nu a.\, a$ are not observationally equivalent terms of type $\mathsf{Name} \to \mathsf{Name}$—see the discussion after Remark 3.2 below; however, they are observationally equivalent in the $\lambda\nu$-calculus.) Even more radically, in Odersky's calculus there is no sharing of local names between the components of a tuple, since $\nu a.\, (t_1\ ,\ t_2)$ is observationally equivalent to $(\nu a_1.\, t_1[a_1/a]\ ,\ \nu a_2.\, t_2[a_2/a])$.

Contribution of this paper. We shed new light on Odersky's version of locally scoped names by showing that it stands in a surprising relation to the more familiar, dynamic allocation interpretation. We prove that Odersky's version of $\nu a.\, t$ provides a 'direct' meaning for locally scoped names from which the behaviour determined by dynamic allocation can be recovered via continuations. More precisely, we show that a standard continuation passing style (CPS) transformation on typed λ-calculus can be extended to locally scoped names so as to provide a computationally adequate translation of ν-calculus into $\lambda\nu$-calculus. Dynamically allocated names at a particular type are translated to Odersky-style local names at the corresponding function type of continuations. Quite surprisingly, even though Odersky's version of $\nu a.\, (-)$ behaves quite differently with respect to functions compared to the dynamic allocation semantics of $\nu a.\, (-)$, we show that the CPS translation is sound and complete for evaluating boolean terms (Theorem 5.1). Since the translation is compositional, it follows that two ν-calculus terms of any type are observationally equivalent if their CPS-translations are observationally equivalent in the $\lambda\nu$-calculus

Our proof of these results is via a new formulation of $\lambda\nu$-calculus 'big step' operational semantics and via a by-now standard use of Felleisen-style evaluation contexts for ν-calculus. At the heart of the proof we construct (in Sect. 5.2) a logical relation between $\lambda\nu$-calculus and ν-calculus tailored to the CPS transformation. Although we use the methods of operational semantics, as we explain in Sect. 6 our results have their origin in a denotational semantics of dynamic allocation using *nominal sets* [18] and, more recently, a simple nominal sets model for Odersky-style local names [14]. Our results suggest re-evaluating the usefulness of Odersky's semantics of locally scoped names and the concluding section gives some avenues for doing that.

$T \in \mathit{Type} ::=$		$t \in \mathit{Term} ::=$	
Name	names	x	variable, $x \in \mathbb{V}$
Bool	booleans	a	atomic name, $a \in \mathbb{A}$
$T \times T$	pairs	$\nu a.\, t$	locally scoped name
$T \to T$	functions	$t = t$	name equality test
		true	truth
		false	falsity
		if t then t else t	conditional
		$(t\ ,\ t)$	pair
		let $(x\ ,\ x) = t$ in t	unpairing
		$\lambda x \to t$	function abstraction
		$t\, t$	application

Figure 1 Syntax

2 Simply Typed λ-Calculus with Local Names

We use the same syntax and typing rules for the Pitts-Stark ν-calculus as for the Odersky $\lambda\nu$-calculus. This unification is just a slight deviation from the original syntax [15, 12], but the expressiveness remains the same. To the usual simply typed λ-calculus with pairs and booleans we add names that can be tested for equality and locally scoped. The types and terms of the resulting language are given in Fig. 1.

It is convenient to use two different sorts of identifier in terms, drawn from disjoint infinite sets $\mathbb{V}$ and $\mathbb{A}$. Elements $x, y, z \ldots$ of $\mathbb{V}$ are called *variables* and elements $a, b, c, \ldots$ of $\mathbb{A}$ are called *atomic names*. We make this syntactic distinction to emphasise the fact that the two different sorts of identifier have different substitution properties. Validity of judgements in the calculi we consider here is preserved under substituting terms for variables; but in general it is only preserved under permutations of atomic names, rather than more general forms of substitution for names.

As a matter of notation we write $t[t_1/x_1, \ldots, t_n/x_n]$ for the (capture-avoiding, simultaneous) substitution of terms $t_1, \ldots, t_n$ for free occurrences of the distinct variables $x_1, \ldots, x_n$ in the term t. We identify terms up to α-equivalence of bound variables and bound atomic names. The binding forms are as follows: free occurrences of a in t become bound in $\nu a.\, t$; free occurrences of x_1 and x_2 in t' become bound in let $(x_1\ ,\ x_2) = t$ in t'; and free occurrences of x in t become bound in $\lambda x \to t$. We write $\mathit{fv}(t)$ and $\mathit{fn}(t)$ respectively for the finite sets of free variables and free atomic names of t. We say that a term t is *variable-closed* if $\mathit{fv}(t) = \emptyset$ (even if $\mathit{fn}(t)$ is non-empty).

The grammar in Fig. 1 specifies 'raw' terms, but we are only interested in well-typed terms. We specify those via an inductively defined typing relation $\Gamma \vdash t : T$, where the *typing context* $\Gamma = \{x_1 : T_1, \ldots, x_n : T_n\}$ is a finite map from variables x_i to types T_i whose domain $\mathit{dom}(\Gamma) = \{x_1, \ldots, x_n\}$ contains the set $\mathit{fv}(t)$ of free variables of t. Rather than also recording the free atomic names of t in the typing context, we have chosen to leave them implicit, because it simplifies notation later. Thus the typing rules involving names are as follows.

$$\frac{}{\Gamma \vdash a : \mathsf{Name}} \qquad \frac{\Gamma \vdash t : T}{\Gamma \vdash \nu a.\, t : T} \qquad \frac{\Gamma \vdash t : \mathsf{Name} \qquad \Gamma \vdash t' : \mathsf{Name}}{\Gamma \vdash t = t' : \mathsf{Bool}}$$

The typing rules for the other syntactic constructs are entirely conventional, so we omit them here.

$$\frac{\overline{a} \cup \{a\} \,,\, t \Downarrow_\nu \overline{a}' \,,\, v}{\overline{a} \,,\, \nu a.\, t \Downarrow_\nu \overline{a}' \,,\, v}\ (a \notin \overline{a}) \qquad \frac{}{\overline{a} \,,\, v \Downarrow_\nu \overline{a} \,,\, v}\ (v = a, \mathsf{true}, \mathsf{false}, \lambda x \to t)$$

$$\frac{\overline{a} \,,\, t_1 \Downarrow_\nu \overline{a}' \,,\, a_1 \qquad \overline{a}' \,,\, t_2 \Downarrow_\nu \overline{a}'' \,,\, a_2}{\overline{a} \,,\, t_1 = t_2 \Downarrow_\nu \overline{a}'' \,,\, \delta_{a_1\, a_2}} \text{ where } \delta_{a_1\, a_2} \triangleq \begin{cases} \mathsf{true} & \text{if } a_1 = a_2 \\ \mathsf{false} & \text{if } a_1 \neq a_2 \end{cases}$$

$$\frac{\overline{a} \,,\, t_1 \Downarrow_\nu \overline{a}' \,,\, \mathsf{true} \qquad \overline{a}' \,,\, t_2 \Downarrow_\nu \overline{a}'' \,,\, v}{\overline{a} \,,\, \mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 \Downarrow_\nu \overline{a}'' \,,\, v} \qquad \frac{\overline{a} \,,\, t_1 \Downarrow_\nu \overline{a}' \,,\, \mathsf{false} \qquad \overline{a}' \,,\, t_3 \Downarrow_\nu \overline{a}'' \,,\, v}{\overline{a} \,,\, \mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 \Downarrow_\nu \overline{a}'' \,,\, v}$$

$$\frac{\overline{a} \,,\, t_1 \Downarrow_\nu \overline{a}' \,,\, v_1 \qquad \overline{a}' \,,\, t_2 \Downarrow_\nu \overline{a}'' \,,\, v_2}{\overline{a} \,,\, (t_1 \,,\, t_2) \Downarrow_\nu \overline{a}'' \,,\, (v_1 \,,\, v_2)} \qquad \frac{\overline{a} \,,\, t \Downarrow_\nu \overline{a}' \,,\, (v_1 \,,\, v_2) \qquad \overline{a}' \,,\, t'[v_1/x_1, v_2/x_2] \Downarrow_\nu \overline{a}'' \,,\, v}{\overline{a} \,,\, \mathsf{let}\ (x_1 \,,\, x_2) = t\ \mathsf{in}\ t' \Downarrow_\nu \overline{a}'' \,,\, v}$$

$$\frac{\overline{a} \,,\, t_1 \Downarrow_\nu \overline{a}' \,,\, \lambda x \to t \qquad \overline{a}' \,,\, t_2 \Downarrow_\nu \overline{a}'' \,,\, v \qquad \overline{a}'' \,,\, t[v/x] \Downarrow_\nu \overline{a}''' \,,\, v'}{\overline{a} \,,\, t_1\, t_2 \Downarrow_\nu \overline{a}''' \,,\, v'}$$

$$\text{where } v \in \mathit{Val} ::= x \mid a \mid \mathsf{true} \mid \mathsf{false} \mid (v \,,\, v) \mid \lambda x \to t$$

Figure 2 ν-Calculus evaluation relation

We will be concerned with various congruence relations between well-typed terms. Here is the general definition of such a relation (cf. [13, Definition 7.5.1]).

▶ **Definition 2.1.** A *type-respecting binary relation* is specified by a set $\mathcal{R}$ of quadruples (Γ, t_1, t_2, T), where $\Gamma \vdash t_1 : T$ and $\Gamma \vdash t_2 : T$. We write $\Gamma \vdash t_1\ \mathcal{R}\ t_2 : T$ instead of $(\Gamma, t_1, t_2, T) \in \mathcal{R}$. Such a relation is a *congruence* if it is reflexive, symmetric, transitive and *compatible* with the term-forming operations. The latter means

$$\Gamma \vdash a\ \mathcal{R}\ a : \mathsf{Name}$$

$$\Gamma \vdash t_1\ \mathcal{R}\ t_2 : T \ \Rightarrow\ \Gamma \vdash \nu a.\, t_1\ \mathcal{R}\ \nu a.\, t_2 : T$$

$$\Gamma \vdash t_1\ \mathcal{R}\ t_2 : \mathsf{Name} \ \wedge\ \Gamma \vdash t : \mathsf{Name} \ \Rightarrow\ \Gamma \vdash (t_1 = t)\ \mathcal{R}\ (t_2 = t) : \mathsf{Bool} \wedge \Gamma \vdash (t = t_1)\ \mathcal{R}\ (t = t_2) : \mathsf{Bool}$$

and similar conditions for the other term-forming operations.

3 ν-Calculus

The language of the previous section becomes the ν-calculus [15, 20] if we evaluate locally scoped names $\nu a.\, t$ using the mechanism of dynamic allocation (and use call-by-value evaluation for pairs and functions). Figure 2 gives rules in the style of the Definition of Standard ML [10] for inductively defining a relation $\overline{a} \,,\, t \Downarrow_\nu \overline{a}' \,,\, v$, where

- $\overline{a}$ and $\overline{a}'$ are finite subsets of $\mathbb{A}$ with $\overline{a} \subseteq \overline{a}'$;
- t and v are variable-closed terms with $\mathit{fn}(t) \subseteq \overline{a}$ and $\mathit{fn}(v) \subseteq \overline{a}'$;
- $v \in \mathit{Val} \subseteq \mathit{Term}$ is a *value*, as specified by the grammar in Fig. 2.

We use this relation to define observational equivalence for the ν-calculus, $\Gamma \vdash t_1 \approx_\nu t_2 : T$. To do so, we believe it is helpful to take the abstract, *relational* point of view first advocated by Gordon and Lassen [8]. We wish $\approx_\nu$ to be a congruence in the sense of Definition 2.1

and to be ν-*adequate* for observing evaluation of boolean terms in the sense that

$$\emptyset \vdash t_1 \approx_\nu t_2 : \mathsf{Bool} \;\wedge\; fn(t_1, t_2) \subseteq \overline{a} \;\Rightarrow\; (\overline{a}\,, t_1 \Downarrow_\nu \_\,, \mathsf{true} \Leftrightarrow \overline{a}\,, t_2 \Downarrow_\nu \_\,, \mathsf{true}) \tag{1}$$

$$\text{where} \quad \overline{a}\,, t \Downarrow_\nu \_\,, \mathsf{true} \;\triangleq\; (\exists \overline{a}')\, \overline{a}\,, t \Downarrow_\nu \overline{a}'\,, \mathsf{true}. \tag{2}$$

▶ **Definition 3.1** (ν-Calculus observational equivalence)**.** Arguing as in the proof of [13, Theorem 7.5.3], we have that the union of all type-respecting binary relations that are both compatible (Definition 2.1) and have the ν-adequacy property (1) is an equivalence relation; and hence it is the largest ν-adequate congruence relation. We denote it by $\approx_\nu$ and call it *ν-calculus observational equivalence*.

The fact that in (1) we observe convergence just to true is not significant; also observing convergence to false, or to a particular atomic name, does not change $\approx_\nu$. On the other hand, just observing convergence *per se* would result in a trivial equivalence, since ν-calculus lacks any non-terminating features such as fixpoint recursion (as a matter of choice rather than necessity).

▶ **Remark 3.2** (contextual equivalence)**.** The terms 'observational equivalence' and 'contextual equivalence' are used more or less interchangeably in the literature on ν-calculus. (One might say that they are contextually equivalent terms.) We have chosen the first, because we favour the more abstract, 'context-free' characterization that we have used as the definition. However, it is possible to give a more concrete characterization of $\approx_\nu$ in terms of substitution of terms into term contexts, that is, syntax-trees with a hole; see [15, Definition 4]. Both free variables and free atomic names in terms may get captured by this form of substitution. So term contexts are not identified up to α-equivalence and one has to give separate and more elaborate typing rules for them. These complications are avoided by using the relational definition we have given.

However one defines it, the properties of $\approx_\nu$ are known to be very complicated; see [2] for a recent discussion of this fact. In particular, terms of function or product type do not behave extensionally up to observational equivalence. For example

$$\emptyset \vdash \nu a.\, \lambda x \to a \not\approx_\nu \lambda x \to \nu a.\, a : \mathsf{Name} \to \mathsf{Name} \tag{3}$$

(since applying $\lambda f \to \nu a.\, (f\, a = f\, a)$ to each term gives terms that evaluate to true and false respectively); and yet applying these two terms to any name yields observationally equivalent results. Similarly

$$\emptyset \vdash \nu a.\, \nu b.\, (a\ ,\ b) \not\approx_\nu \nu a.\, (a\ ,\ a) : \mathsf{Name} \times \mathsf{Name} \tag{4}$$

(since applying $\lambda x \to \mathsf{let}\ (x_1\ ,\ x_2) = x\ \mathsf{in}\ (x_1 = x_2)$ to each term gives terms that evaluate to false and true respectively); and yet applying first and second projection functions to them yields observationally equivalent results in each case.

4 $\lambda\nu$-Calculus

Figure 3 inductively defines a state-free evaluation relation $t \Downarrow_{\lambda\nu} c$, where t and c are variable-closed terms (possibly with free atomic names) and c is a *canonical form*, that is, in the subset $Cf \subseteq Term$ of terms specified by the grammar at the bottom of the figure. The rules for evaluating booleans, pairs and functions are just those of the pure call-by-name typed λ-calculus. It is the first rule in the figure, for evaluating $\nu a.\, t$, that embodies

$$\frac{t \Downarrow_{\lambda\nu} c}{\nu a.\, t \Downarrow_{\lambda\nu} a\!\setminus\! c} \text{ where } a\!\setminus\! c \triangleq \begin{cases} \nu a.\, a & \text{if } c = a \\ c & \text{if } c \in (\mathbb{A} - \{a\}) \cup \{\nu a.\, a, \mathsf{true}, \mathsf{false}\} \\ (\nu a.\, t_1 \ , \nu a.\, t_2) & \text{if } c = (t_1 \ , t_2) \\ \lambda x \to \nu a.\, t & \text{if } c = \lambda x \to t \end{cases}$$

$$\frac{}{c \Downarrow_{\lambda\nu} c} \qquad \frac{t_1 \Downarrow_{\lambda\nu} c_1 \qquad t_2 \Downarrow_{\lambda\nu} c_2}{t_1 = t_2 \Downarrow_{\lambda\nu} \delta_{c_1\, c_2}} \text{ where } \delta_{c_1\, c_2} \triangleq \begin{cases} \mathsf{true} & \text{if } c_1 = c_2 \\ \mathsf{false} & \text{if } c_1 \neq c_2 \end{cases}$$

$$\frac{t_1 \Downarrow_{\lambda\nu} \mathsf{true} \qquad t_2 \Downarrow_{\lambda\nu} c}{\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 \Downarrow_{\lambda\nu} c} \qquad \frac{t_1 \Downarrow_{\lambda\nu} \mathsf{false} \qquad t_3 \Downarrow_{\lambda\nu} c}{\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3 \Downarrow_{\lambda\nu} c}$$

$$\frac{t \Downarrow_{\lambda\nu} (t_1 \ , t_2) \qquad t'[t_1/x_1, t_2/x_2] \Downarrow_{\lambda\nu} c}{\mathsf{let}\ (x_1 \ , x_2) = t\ \mathsf{in}\ t' \Downarrow_{\lambda\nu} c} \qquad \frac{t_1 \Downarrow_{\lambda\nu} \lambda x \to t \qquad t[t_2/x] \Downarrow_{\lambda\nu} c}{t_1\, t_2 \Downarrow_{\lambda\nu} c}$$

$$\text{where } c \in Cf ::= a \mid \nu a.\, a \mid \mathsf{true} \mid \mathsf{false} \mid (t \ , t) \mid \lambda x \to t$$

Figure 3 $\lambda\nu$-Calculus evaluation relation

Odersky's semantics of locally scoped names from [12]. Compared with the corresponding rule in Fig. 2, whose effect is to extrude local scopes outward to the top level, here scoping intrudes through pairing and function abstraction until it reaches canonical booleans and names.

▶ **Remark 4.1** (the 'anonymous name' anon $\triangleq \nu a.\, a$)**.** What we here call the $\lambda\nu$-calculus is essentially the typed calculus described in [12, Sect. 6] equipped with the deterministic evaluation relation sketched in Sect. 4 of that paper (although our description of evaluation in Fig. 3 is more direct). However, there are two related respects in which our calculus differs. Firstly, we choose to regard the term anon $\triangleq \nu a.\, a$ as a canonical form of type Name and secondly, we take the boolean term anon = anon to evaluate to true. Whereas Odersky takes both terms to be stuck with respect to evaluation (and to be bottom, denotationally). Other choices are possible; for example one might take anon to be canonical, but have anon = anon evaluate to false, or be stuck. Such choices clearly affect the properties of $\lambda\nu$-calculus contextual equivalence and hence potentially affect the adequacy of translations of ν-calculus into $\lambda\nu$-calculus that we develop in the next section.

Our motivation for taking anon to be a canonical form comes from the nominal sets model of Odersky-style local names described in [14], where anon is a non-bottom value. Having stuck terms, Odersky's original typed system fails to satisfy the usual 'progress' part of type soundness, whereas here we have the following result.

▶ **Theorem 4.2** ($\lambda\nu$-calculus type soundness and totality)**.** *In the $\lambda\nu$-calculus, well-typed variable-closed terms possess unique canonical forms: for all $\emptyset \vdash t : T$, there is a unique c satisfying $\emptyset \vdash c : T$ and $t \Downarrow_{\lambda\nu} c$.*

Proof. The proof that evaluation preserves typing is routine. That evaluation is single-valued follows from the fact that it does not create free atomic names ($t \Downarrow_{\lambda\nu} c \Rightarrow fn(c) \subseteq fn(t)$); this follows in turn from the fact that in the derived operation $a\!\setminus\! c$ on canonical forms used to evaluate $\nu a.\, t$, free occurrences of a in c become bound in $a\!\setminus\! c$. Finally one has to prove that evaluation of well-typed terms is total. This can be done by adapting the

usual argument for simply typed λ-calculus using Tait-style computability predicates; we omit the details here. ◀

As for the ν-calculus, we can give a simple, 'relational' definition of observational equivalence.

▶ **Definition 4.3** ($\lambda\nu$-Calculus observational equivalence)**.** We define $\approx_{\lambda\nu}$, to be the largest congruence relation satisfying the following *$\lambda\nu$-adequacy property*:

$$\emptyset \vdash t_1 \approx_{\lambda\nu} t_2 : \mathsf{Bool} \;\Rightarrow\; (t_1 \Downarrow_{\lambda\nu} \mathsf{true} \;\Leftrightarrow\; t_2 \Downarrow_{\lambda\nu} \mathsf{true}). \tag{5}$$

It can be constructed by observing that the union of all $\lambda\nu$-adequate and compatible type-respecting relations is an equivalence relation (as well as being $\lambda\nu$-adequate and compatible).

Modulo the changes mentioned in Remark 4.1, $\approx_{\lambda\nu}$ is essentially the same notion that Odersky defines more concretely with term contexts [12, Sect. 5]. He shows that it has many pleasant properties in common with the pure typed λ-calculus, such as extensionality for functions and products. One can show that

$$\emptyset \vdash t : T \;\wedge\; t \Downarrow_{\lambda\nu} c \;\Rightarrow\; \emptyset \vdash t \approx_{\lambda\nu} c : T \tag{6}$$

$$\Gamma \vdash t : T \;\wedge\; a \notin \mathit{fn}(t) \;\Rightarrow\; \Gamma \vdash \nu a.\, t \approx_{\lambda\nu} t : T \tag{7}$$

$$\Gamma \vdash t : T \;\Rightarrow\; \Gamma \vdash \nu a.\, \nu a'.\, t \approx_{\lambda\nu} \nu a'.\, \nu a.\, t : T. \tag{8}$$

Hence in particular the pairs of terms in (3) and (4) are observationally equivalent in the $\lambda\nu$-calculus. So $\approx_\nu$ and $\approx_{\lambda\nu}$ are not at all the same. Indeed in view of property (6), the evaluation rules in Fig. 3 imply the characteristic 'scope intrusion' laws

$$\Gamma \vdash \nu a.\, \lambda x \to t \approx_{\lambda\nu} \lambda x \to \nu a.\, t : T_1 \to T_2 \tag{9}$$

$$\Gamma \vdash \nu a.\, (t_1 \,,\, t_2) \approx_{\lambda\nu} (\nu a.\, t_1 \,,\, \nu a.\, t_2) : T_1 \times T_2 \tag{10}$$

that distinguish Odersky-style local names from dynamically allocated ones.

5 Translating ν to $\lambda\nu$

Figure 4 gives a *continuation-passing style* (CPS) transformation of the types and terms of the typed λ-calculus from Sect. 2. The transformations for values ($v \mapsto v^\bullet$) and for terms ($t \mapsto t^\circ$) are defined by mutual recursion on the structure of these expressions.

The part of the transformation that does not concern local names is very standard: we have combined Moggi's call-by-value translation of λ-calculus into his computational metalanguage [11] with an interpretation of that metalanguage that uses the continuation monad $\mathcal{C}(-) \triangleq (- \to \mathsf{Bool}) \to \mathsf{Bool}$. The part of the transformation that *does* concern local names is pleasingly simple; *dynamically allocated local names at a type T are transformed into Odersky-style local names at type $\mathcal{C}\overline{T}$*: $(\nu a.\, t)^\circ = \nu a.\, t^\circ$.

Recalling the definitions of $\Downarrow_\nu$ and $\approx_\nu$ for the ν-calculus from Sect. 3 and $\Downarrow_{\lambda\nu}$ and $\approx_{\lambda\nu}$ for the $\lambda\nu$-calculus from Sect. 4, we can now state the main result of the paper.

▶ **Theorem 5.1** (computational adequacy)**.** *(i) For all $\emptyset \vdash t : \mathsf{Bool}$, with $\mathit{fn}(t) \subseteq \overline{a}$ say,*

$$\overline{a}\,, t \Downarrow_\nu \_\,, \mathsf{true} \;\Leftrightarrow\; t^\circ(\lambda x \to x) \Downarrow_{\lambda\nu} \mathsf{true}. \tag{11}$$

(ii) For all $\Gamma \vdash t_i : T$ $(i = 1, 2)$, if $\overline{\Gamma} \vdash t_1^\circ \approx_{\lambda\nu} t_2^\circ : \mathcal{C}\overline{T}$, then $\Gamma \vdash t_1 \approx_\nu t_2 : T$.

- Types $\mapsto$ types $\overline{T}$
$$\left\{\begin{array}{rcl} \overline{\mathsf{Name}} & = & \mathsf{Name} \\ \overline{\mathsf{Bool}} & = & \mathsf{Bool} \\ \overline{T_1 \times T_2} & = & \overline{T_1} \times \overline{T_2} \\ \overline{T_1 \to T_2} & = & \overline{T_1} \to \mathcal{C}\overline{T_2} \\ \text{where} \quad \mathcal{C}T & \triangleq & (T \to \mathsf{Bool}) \to \mathsf{Bool}. \end{array}\right.$$
- Typing contexts $\Gamma \mapsto$ typing contexts $\overline{\Gamma}$
$$\left\{\begin{array}{rcl} \overline{\emptyset} & = & \emptyset \\ \overline{\Gamma, x : T} & = & \overline{\Gamma}, x : \overline{T}. \end{array}\right.$$
- Values $\Gamma \vdash v : T \mapsto$ canonical forms $\overline{\Gamma} \vdash v^\bullet : \overline{T}$
$$\begin{aligned} v^\bullet &= v \quad \text{for } v = x, a, \mathsf{true}, \mathsf{false} \\ (v_1 \mathbin{,} v_2)^\bullet &= (v_1^\bullet \mathbin{,} v_2^\bullet) \\ (\lambda x_1 \to t_1)^\bullet &= \lambda x_1 \to t_1^\circ. \end{aligned}$$
- Terms $\Gamma \vdash t : T \mapsto$ terms $\overline{\Gamma} \vdash t^\circ : \mathcal{C}\overline{T}$
$$\begin{aligned} v^\circ &= \lambda k \to k\, v^\bullet \\ (\nu a.\, t)^\circ &= \nu a.\, t^\circ \\ (t_1 = t_2)^\circ &= \lambda k \to t_1^\circ (\lambda x \to t_2^\circ (\lambda x' \to \mathsf{if}\ x = x'\ \mathsf{then}\ k\, \mathsf{true}\ \mathsf{else}\ k\, \mathsf{false})) \\ (\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3)^\circ &= \lambda k \to t_1^\circ (\lambda x \to \mathsf{if}\ x\ \mathsf{then}\ t_2^\circ\, k\ \mathsf{else}\ t_3^\circ\, k) \\ (t_1 \mathbin{,} t_2)^\circ &= \lambda k \to t_1^\circ (\lambda x \to t_2^\circ (\lambda x' \to k\,(x \mathbin{,} x'))) \quad \text{when } (t_1 \mathbin{,} t_2) \notin \mathit{Val} \\ (\mathsf{let}\ (x_1 \mathbin{,} x_2) = t_1\ \mathsf{in}\ t_2)^\circ &= \lambda k \to t_1^\circ (\lambda x \to \mathsf{let}\ (x_1 \mathbin{,} x_2) = x\ \mathsf{in}\ t_2^\circ\, k) \\ (t_1\, t_2)^\circ &= \lambda k \to t_1^\circ (\lambda x \to t_2^\circ (\lambda x' \to x\, x'\, k)) \\ (\text{where } k, x, x' \notin \mathit{fv}(v, t_1, t_2, t_3)). & \end{aligned}$$

Figure 4 CPS transformation

Part (ii) of the theorem follows from part (i), because the CPS transformation is compositional. More precisely, referring to Definition 2.1, the type-respecting binary relation

$$\mathcal{R} \triangleq \{(\Gamma, t_1, t_2, T) \mid \Gamma \vdash t_1 : T \ \wedge\ \Gamma \vdash t_2 : T \ \wedge\ \overline{\Gamma} \vdash t_1^\circ \approx_{\lambda\nu} t_2^\circ : \mathcal{C}\overline{T}\}$$

is easily seen to be a congruence; additionally it has the ν-adequacy property (1) by virtue of (i) and because $\approx_{\lambda\nu}$ is a $\lambda\nu$-adequate congruence. So since $\approx_\nu$ is by definition the largest ν-adequate congruence, it contains $\mathcal{R}$—as required for property (ii).

The rest of this section sketches the proof of part (i) of the theorem. We first re-formulate the operational semantics of the ν-calculus in terms of an abstract machine with frame stacks. As a result, property (11) becomes a statement about machine configurations with an empty stack that can be deduced from a more general bi-implication involving arbitrary frame stacks (Corollary 5.8). The left-to-right part of this bi-implication is straightforward; the right-to-left part is harder and we prove it by constructing a suitable logical relation between the $\lambda\nu$-calculus and the ν-calculus.

5.1 Abstract machine

Although the 'big step' operational semantics of Sect. 3 gives a clear specification of the ν-calculus, experience has shown that a small-step semantics formulated in the style of Felleisen with evaluation contexts [5] is better suited for developing the properties of the associated observational equivalence, $\approx_\nu$; and to make proofs about evaluation contexts easier to formalize, it pays to write them 'inside out' as a list of basic contexts (evaluation

$$
\begin{aligned}
\langle F\,,\,\nu a.\,t\rangle &\to_\nu \langle F\,,\,t\rangle \quad \text{if } a \notin \mathit{fn}(F)\\
\langle F\,,\,t_1 = t_2\rangle &\to_\nu \langle F \circ (\cdot = t_2)\,,\,t_1\rangle\\
\langle F\,,\,\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3\rangle &\to_\nu \langle F \circ (\mathsf{if}\ \cdot\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3)\,,\,t_1\rangle\\
\langle F\,,\,(t_1\ ,\ t_2)\rangle &\to_\nu \langle F \circ (\cdot\ ,\ t_2)\,,\,t_1\rangle \quad \text{when } (t_1\ ,\ t_2) \notin \mathit{Val}\\
\langle F\,,\,\mathsf{let}\ (x_1\ ,\ x_2) = t\ \mathsf{in}\ t'\rangle &\to_\nu \langle F \circ (\mathsf{let}\ (x_1\ ,\ x_2) = \cdot\ \mathsf{in}\ t')\,,\,t\rangle\\
\langle F\,,\,t_1\,t_2\rangle &\to_\nu \langle F \circ (\cdot\,t_2)\,,\,t_1\rangle\\
\langle F \circ (\cdot = t)\,,\,a\rangle &\to_\nu \langle F \circ (a = \cdot)\,,\,t\rangle\\
\langle F \circ (a_1 = \cdot)\,,\,a_2\rangle &\to_\nu \langle F\,,\,\delta_{a_1\,a_2}\rangle\\
\langle F \circ (\mathsf{if}\ \cdot\ \mathsf{then}\ t\ \mathsf{else}\ t')\,,\,\mathsf{true}\rangle &\to_\nu \langle F\,,\,t\rangle\\
\langle F \circ (\mathsf{if}\ \cdot\ \mathsf{then}\ t\ \mathsf{else}\ t')\,,\,\mathsf{false}\rangle &\to_\nu \langle F\,,\,t'\rangle\\
\langle F \circ (\cdot\ ,\ t)\,,\,v\rangle &\to_\nu \langle F \circ (v\ ,\ \cdot)\,,\,t\rangle\\
\langle F \circ (v_1\ ,\ \cdot)\,,\,v_2\rangle &\to_\nu \langle F\,,\,(v_1\ ,\ v_2)\rangle\\
\langle F \circ (\mathsf{let}\ (x_1\ ,\ x_2) = \cdot\ \mathsf{in}\ t)\,,\,(v_1\ ,\ v_2)\rangle &\to_\nu \langle F\,,\,t[v_1/x_1.v_2/x_2]\rangle\\
\langle F \circ (\cdot\,t)\,,\,v\rangle &\to_\nu \langle F \circ (v\,\cdot)\,,\,t\rangle\\
\langle F \circ (\lambda x \to t)\,\cdot\,,\,v\rangle &\to_\nu \langle F\,,\,t[v/x]\rangle
\end{aligned}
$$

where $F \in \mathit{Stack} ::= \mathsf{Id} \mid F \circ E$ and
$E \in \mathit{Frame} ::= \cdot = t \mid v = \cdot \mid \mathsf{if}\ \cdot\ \mathsf{then}\ t\ \mathsf{else}\ t \mid (\cdot\ ,\ t) \mid (v\ ,\ \cdot) \mid \mathsf{let}\ (x\ ,\ x) = \cdot\ \mathsf{in}\ t \mid \cdot\,t \mid v\,\cdot$

Figure 5 ν-Calculus abstract machine

frames). Figure 5 formulates the operational semantics of the ν-calculus in this style. It defines a binary relation $\to_\nu$ between configurations of the form $\langle F\,,\,t\rangle$, where

- F is a *frame stack* (a list of *evaluation frames* E, defined by the grammar in the figure);
- t is a term;
- both F and t are variable-closed.

Note the first transition in Fig. 5, for dynamically allocated local names. The use of sets of atomic names $\overline{a}$ as states in the definition of $\Downarrow_\nu$ is not necessary for $\to_\nu$; the implicit state of a configuration $\langle F\,,t\rangle$ is its finite set $\mathit{fn}(F) \cup \mathit{fn}(t)$ of free atomic names. The termination relation (2) used in the definition of $\approx_\nu$ can be characterized in terms of termination of the abstract machine, as follows.

▶ **Lemma 5.2.** *Let t be a variable-closed term, with $\mathit{fn}(t) \subseteq \overline{a}$ say. Then $\overline{a}\,,t \Downarrow_\nu \_\,,\mathsf{true}$ holds iff $\langle \mathsf{Id}\,,t\rangle \to_\nu^* \langle \mathsf{Id}\,,\mathsf{true}\rangle$, where $\to_\nu^*$ denotes the reflexive-transitive closure of $\to_\nu$.*

Proof. For the left-to-right implication, one can show (by induction on the derivation from the rules in Fig. 2) that $\overline{a}\,,t \Downarrow_\nu \overline{a}'\,,v$ implies $(\forall F)\ \mathit{fn}(F) \cap \overline{a}' = \emptyset \ \Rightarrow\ \langle F\,,t\rangle \to_\nu^* \langle F\,,v\rangle$. The right-to-left implication can be deduced from

$$\langle F\,,t\rangle \to_\nu \langle F'\,,t'\rangle \ \wedge\ \mathit{fn}(F,t,F',t') \subseteq \overline{a} \ \wedge\ \overline{a}\,,F'[t'] \Downarrow_\nu \_\,,v \ \Rightarrow\ \overline{a}\,,F[t] \Downarrow_\nu \_\,,v \qquad (12)$$

where the term $F[t]$ is defined by recursion on the length of the frame stack F

$$\mathsf{Id}[t] = t \quad \text{and} \quad (F \circ E)[t] = F[E[t/\cdot]] \qquad (13)$$

and where $E[t/\cdot]$ is the term obtained from an evaluation frame E by replacing its hole $\cdot$ by the term t. Property (12) is proved by case analysis on the definition of $\to_\nu$ in Fig. 5, using

$$\overline{a}\,,F[t] \Downarrow_\nu \overline{a}'\,,v \ \Leftrightarrow\ (\exists \overline{a}'', v')\ \overline{a}\,,t \Downarrow_\nu \overline{a}''\,,v' \ \wedge\ \overline{a}''\,,F[v'] \Downarrow_\nu \overline{a}'\,,v$$

- Frame stacks $\Gamma \vdash F : T \to \mathsf{Bool} \mapsto$ canonical forms $\overline{\Gamma} \vdash F^* : \overline{T} \to \mathsf{Bool}$

$$
\begin{aligned}
\mathsf{Id}^* &= \lambda x \to x \\
(F \circ (\cdot = t_2))^* &= \lambda x \to t_2^\circ(\lambda x' \to \mathsf{if}\ x = x'\ \mathsf{then}\ F^*\mathsf{true}\ \mathsf{else}\ F^*\mathsf{false}) \\
(F \circ (v_1 = \cdot))^* &= \lambda x \to \mathsf{if}\ v_1^\bullet = x\ \mathsf{then}\ F^*\mathsf{true}\ \mathsf{else}\ F^*\mathsf{false} \\
(F \circ (\mathsf{if}\ \cdot\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2))^* &= \lambda x \to \mathsf{if}\ x\ \mathsf{then}\ t_1^\circ F^*\ \mathsf{else}\ t_2^\circ F^* \\
(F \circ (\cdot\ ,\ t_2))^* &= \lambda x \to t_2^\circ(\lambda x' \to F^*(x\ ,\ x')) \\
(F \circ (v_1\ ,\ \cdot))^* &= \lambda x \to F^*(v_1^\bullet\ ,\ x) \\
(F \circ (\mathsf{let}\ (x_1\ ,\ x_2) = \cdot\ \mathsf{in}\ t))^* &= \lambda x \to \mathsf{let}\ (x_1\ ,\ x_2) = x\ \mathsf{in}\ t^\circ F^* \\
(F \circ (\cdot\ t_2))^* &= \lambda x \to t_2^\circ(\lambda x' \to x\ x'\ F^*) \\
(F \circ (v_1\ \cdot))^* &= \lambda x \to v_1^\bullet\ x\ F^* \\
\text{(where } x, x' \notin fv(v_1, t_1, t_2, t, F) &\text{ and } x_1, x_2 \notin fv(F))
\end{aligned}
$$

Figure 6 CPS transformation for frame stacks

which in turn is proved by induction on the length of F. ◀

▶ **Definition 5.3** (typed frame stacks)**.** We use the typing relation for terms from Sect. 2 to type ν-calculus frame stacks by substituting a fresh variable for the hole. Thus we write $\Gamma \vdash F : T' \to T$ to mean that $\Gamma, x : T' \vdash F[x] : T$ holds for some/any $x \notin dom(\Gamma)$. (An equivalent, syntax-directed inductive definition of $\Gamma \vdash F : T' \to T$ is of course possible.)

▶ **Notation 5.4.** For each type $T \in \mathit{Type}$, we write $\mathit{Term}(T)$ for the variable-closed terms of type t, that is, those $t \in \mathit{Term}$ satisfying $\emptyset \vdash t : T$. (Note that such a t may have free atomic names.) Similarly $\mathit{Val}(T)$ and $\mathit{Stack}(T' \to T)$ denote the sets of variable-closed ν-calculus values and frame stacks of types T and $T' \to T$ respectively. We define $\mathit{Config}(T) \triangleq \{\langle F\,, t\rangle \mid (\exists\, T' \in \mathit{Type})\ F \in \mathit{Stack}(T' \to T)\ \wedge\ t \in \mathit{Term}(T')\}$.

The CPS transformation for terms can be extended to frame stacks. This is done in Fig. 6 and the next lemma proves the soundness of the transformation.

▶ **Lemma 5.5** (soundness of the CPS transformation)**.** *For each* $\langle F\,, t\rangle \in \mathit{Config}(\mathsf{Bool})$*, if* $\langle F\,, t\rangle \to_\nu^* \langle \mathsf{Id}\,, \mathsf{true}\rangle$ *then* $t^\circ\ F^* \Downarrow_{\lambda\nu} \mathsf{true}$*.*

Proof. Note that $\mathsf{true}^\circ\ \mathsf{Id}^* = (\lambda k \to k\,\mathsf{true})\,(\lambda x \to x) \Downarrow_{\lambda\nu} \mathsf{true}$. So it suffices to show that if $\langle F\,, t\rangle \to_\nu \langle F'\,, t'\rangle$ and $t'^\circ\ F'^* \Downarrow_{\lambda\nu} \mathsf{true}$, then $t^\circ\ F^* \Downarrow_{\lambda\nu} \mathsf{true}$. This can be proved by case analysis on the definition of $\to_\nu$ in Fig. 5. For the two cases in that figure involving substitution of values for variables one first needs to show $(t[v/x])^\circ = t^\circ[v^\bullet/x]$, which can be done by induction on the structure of t. ◀

5.2 Logical relation

To prove the converse of Lemma 5.5 we use the following logical relation between the $\lambda\nu$-calculus and the ν-calculus.

▶ **Definition 5.6.** The relation $t' \blacktriangleleft v : T$, where $T \in \mathit{Type}$, $v \in \mathit{Val}(T)$ and $t' \in \mathit{Term}(\overline{T})$, is defined by recursion on the structure of types T, making use of auxiliary relations $\triangleleft$ and

$\lhd^*$ for terms and frame stacks that are defined in terms of $\blacktriangleleft$:

$$\begin{aligned}
t' \blacktriangleleft a : \mathsf{Name} &\;\Leftrightarrow\; t' \Downarrow_{\lambda\nu} a \\
t' \blacktriangleleft b : \mathsf{Bool} &\;\Leftrightarrow\; t' \Downarrow_{\lambda\nu} b \quad (b \in \{\mathsf{true}, \mathsf{false}\}) \\
t' \blacktriangleleft (v_1 \,,\, v_2) : T_1 \times T_2 &\;\Leftrightarrow\; (\forall t_1, t_2)\; t' \Downarrow_{\lambda\nu} (t_1 \,,\, t_2) \;\Rightarrow\; t_1 \blacktriangleleft v_1 : T_1 \;\wedge\; t_2 \blacktriangleleft v_2 : T_2 \\
t' \blacktriangleleft v : T_1 \to T_2 &\;\Leftrightarrow\; (\forall t_1, v_1)\; t_1 \blacktriangleleft v_1 : T_1 \;\Rightarrow\; t'\, t_1 \lhd v\, v_1 : T_2
\end{aligned}$$

where for $T \in \mathit{Type}$, $t \in \mathit{Term}(T)$, $t' \in \mathit{Term}(\mathcal{C}\overline{T})$, $F \in \mathit{Stack}(T \to \mathsf{Bool})$ and $t'' \in \mathit{Term}(\overline{T} \to \mathsf{Bool})$ we define

$$\begin{aligned}
t' \lhd t : T &\;\triangleq\; (\forall t_1, F)\; t_1 \lhd^* F : T \to \mathsf{Bool} \;\Rightarrow\; t'\, t_1 \Downarrow_{\lambda\nu} \mathsf{true} \;\Rightarrow\; \langle F \,,\, t\rangle \to_\nu^* \langle \mathsf{Id} \,,\, \mathsf{true}\rangle \\
t'' \lhd^* F : T \to \mathsf{Bool} &\;\triangleq\; (\forall t_1, v)\; t_1 \blacktriangleleft v : T \;\Rightarrow\; t''\, t_1 \Downarrow_{\lambda\nu} \mathsf{true} \;\Rightarrow\; \langle F \,,\, v\rangle \to_\nu^* \langle \mathsf{Id} \,,\, \mathsf{true}\rangle.
\end{aligned}$$

The relation $\blacktriangleleft$ is extended to substitutions:

$$\Gamma \vdash \rho \blacktriangleleft \sigma \;\triangleq\; (\forall x \in \mathit{dom}(\Gamma))\; \rho(x) \blacktriangleleft \sigma(x) : \Gamma(x)$$

where ρ (respectively σ) ranges over finite functions from variables to variable-closed terms (respectively variable-closed values). Finally we extend the relations to open values, terms and frame stacks:

$$\begin{aligned}
\Gamma \vdash t' \blacktriangleleft v : T &\;\triangleq\; \overline{\Gamma} \vdash t' : \overline{T} \;\wedge\; \Gamma \vdash v : T \;\wedge\; (\forall \rho, \sigma)\; \Gamma \vdash \rho \blacktriangleleft \sigma \;\Rightarrow\; t'[\rho] \blacktriangleleft v[\sigma] : T \\
\Gamma \vdash t' \lhd t : T &\;\triangleq\; \overline{\Gamma} \vdash t' : \overline{T} \;\wedge\; \Gamma \vdash t : T \;\wedge\; (\forall \rho, \sigma)\; \Gamma \vdash \rho \blacktriangleleft \sigma \;\Rightarrow\; t'[\rho] \lhd t[\sigma] : T \\
\Gamma \vdash t' \lhd^* F : T \to \mathsf{Bool} &\;\triangleq\; \overline{\Gamma} \vdash t' : \overline{T} \to \mathsf{Bool} \;\wedge\; \Gamma \vdash F : T \to \mathsf{Bool} \;\wedge \\
&\qquad (\forall \rho, \sigma)\; \Gamma \vdash \rho \blacktriangleleft \sigma \;\Rightarrow\; t'[\rho] \lhd^* F[\sigma] : T \to \mathsf{Bool}
\end{aligned}$$

▶ **Theorem 5.7** (fundamental property of the logical relation)**.**

$$\Gamma \vdash v : T \;\Rightarrow\; \Gamma \vdash v^\bullet \blacktriangleleft v : T \tag{14}$$

$$\Gamma \vdash t : T \;\Rightarrow\; \Gamma \vdash t^\circ \lhd t : T \tag{15}$$

$$\Gamma \vdash F : T \to \mathsf{Bool} \;\Rightarrow\; \Gamma \vdash F^* \lhd^* F : T \to \mathsf{Bool}. \tag{16}$$

Proof (sketch). Properties (14) and (15) are proved simultaneously by induction on the structure of v and t; and then (16) follows by induction on the structure of F. Here we give just the induction step for the case of locally scoped names; and for this it suffices to show that $t' \lhd t : T$ implies $\nu a.\, t' \lhd \nu a.\, t : T$ So suppose

$$t' \lhd t : T. \tag{17}$$

Referring to the definition of $\lhd$ in terms of $\lhd^*$ in Definition 5.6, we have to show that if

$$t_1 \lhd^* F : T \to \mathsf{Bool} \tag{18}$$

$$(\nu a.\, t')t_1 \Downarrow_{\lambda\nu} \mathsf{true} \tag{19}$$

then $\langle F \,,\, \nu a.\, t\rangle \to_\nu^* \langle \mathsf{Id} \,,\, \mathsf{true}\rangle$. Since we identify terms up to α-equivalence of bound atomic names, we may assume $a \notin \mathit{fn}(t_1, F)$. It follows from the definition of $\Downarrow_{\lambda\nu}$ in Fig. 3 that (19) implies $t't_1 \Downarrow_{\lambda\nu} \mathsf{true}$, since $a \notin \mathit{fn}(t_1)$. From this, (17) and (18), the definition of $\lhd$ gives us $\langle F \,,\, t\rangle \to_\nu^* \langle \mathsf{Id} \,,\, \mathsf{true}\rangle$. Then since $a \notin \mathit{fn}(F)$, from the definition of $\to_\nu$ in Fig. 5 we get $\langle F \,,\, \nu a.\, t\rangle \to_\nu^* \langle \mathsf{Id} \,,\, \mathsf{true}\rangle$, as required. ◀

▶ **Corollary 5.8.** *If* $\langle F\,, t\rangle \in \mathit{Config}(\mathsf{Bool})$*, then*

$$\langle F\,, t\rangle \to_\nu^* \langle \mathsf{Id}\,, \mathsf{true}\rangle \;\Leftrightarrow\; t^\circ F^* \Downarrow_{\lambda\nu} \mathsf{true}. \tag{20}$$

Proof. The left-to-right implication in (20) is the soundness Lemma 5.5. For the converse, since $\langle F\,, t\rangle \in \mathit{Config}(\mathsf{Bool})$, we have $t \in \mathit{Term}(T)$ and $F \in \mathit{Stack}(T \to \mathsf{Bool})$ for some $T \in \mathit{Type}$. Note that by the fundamental property of the logical relation (Theorem 5.7) we have $t^\circ \lhd t : T$ and $F^* \lhd^* F : T \to \mathsf{Bool}$. Then the right-to-left implication follows immediately from the definition of $\lhd$ in terms of $\lhd^*$ in Definition 5.6. ◀

We can now complete the proof of the main theorem.

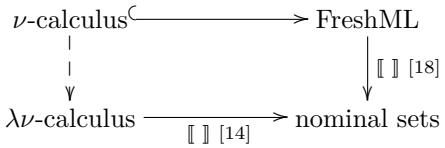
Proof of Theorem 5.1. We have already noted how part (ii) of the theorem follows from part (i). For the latter, combine Lemma 5.2 with the special case of Corollary 5.8 when $F = \mathsf{Id}$, for which $F^* = \mathsf{Id}^* = \lambda x \to x$. ◀

6 A Denotational Perspective

The results in this paper have two sources of inspiration.

- The FreshML language [19], which adds to an ML-like language facilities for declaring and computing with data involving name-binding operations. The 'fresh' in FreshML refers to the fact that the language's mechanism for computing with bound names involves dynamic allocation of fresh names. FreshML's type system ensures that even though programmers have access to the names of bound entities, α-renamed variants of data are indistinguishable up to observational equivalence in the language. This is proved in [18] via a denotational semantics of FreshML (and hence of dynamically allocated local names) using nominal sets [7].
- A nominal sets semantics for Odersky-style locally scoped names given by Pitts in connection with his work on structural recursion modulo α-equivalence [14].

In retrospect, one can see that the denotational semantics in [18] uses a continuation monad in order for the denotation of types to be valued in the 'nominal restriction sets' of [14, Sect. 2.3], rather than just in nominal sets; the restriction operation is then used to interpret locally scoped names. Thus the following picture emerges.



The dotted arrow is the syntactic translation of ν-calculus into $\lambda\nu$-calculus that we have developed in this paper. It composes with the denotational semantics in [14] to recover that in [18] when restricted to the sub-language of FreshML consisting of the ν-calculus.

This suggests an alternative approach to the main result, Theorem 5.1. Instead of the direct, operationally-based proof we have given, one could define a denotational semantics of $\lambda\nu$-calculus using nominal sets, as in [14]. Composing with the CPS transformation gives a denotational semantics for ν-calculus which can be proved adequate for $\Downarrow_\nu$ by constructing a logical relation between semantics and syntax along the lines of that in [18, Sect. 3]. The right-to-left implication in (20) follows from this adequacy result and hence we get an alternative, albeit less direct, proof of the main theorem.

7 Failure of Full Abstraction

Theorem 5.1(ii) says that the CPS translation of ν-calculus into $\lambda\nu$-calculus reflects observational equivalence. If a language translation not only reflects observational equivalence but also preserves it, then one says that the translation is *fully abstract* (by analogy with the use of that terminology for a denotational semantics). For this property to hold, roughly speaking the target language must not be able to observe more about a translated term than is possible in the source language. This is certainly not the case for the CPS translation we have used in this paper. For example, it follows from the theory in [15] that in the ν-calculus the values $v_1 \triangleq \lambda f \to (\lambda x \to \mathsf{true})(f\,\mathsf{true})$ and $v_2 \triangleq \lambda f \to \mathsf{true}$ satisfy $\emptyset \vdash v_1 \approx_\nu v_2 : (\mathsf{Bool} \to \mathsf{Bool}) \to \mathsf{Bool}$. However, in the $\lambda\nu$-calculus one has

$$\emptyset \vdash v_1^\circ \not\approx_{\lambda\nu} v_2^\circ : \mathcal{C}\overline{(\mathsf{Bool} \to \mathsf{Bool}) \to \mathsf{Bool}}$$

because one can calculate from the definition in Fig. 3 that $v_1^\circ(\lambda f \to f\,F\,T) \Downarrow_{\lambda\nu} \mathsf{false}$ and $v_2^\circ(\lambda f \to f\,F\,T) \Downarrow_{\lambda\nu} \mathsf{true}$, where $F \triangleq \lambda x \to \lambda k \to \mathsf{false} \in \mathit{Term}(\overline{\mathsf{Bool} \to \mathsf{Bool}})$ and $T \triangleq \lambda x \to \mathsf{true} \in \mathit{Term}(\mathsf{Bool} \to \mathsf{Bool})$. (For simplicity we have used a pair of values whose equivalence depends upon the absence of non-terminating features in the ν-calculus; more complicated counter-examples exist if one adds recursion to the calculi.)

Note that these evaluations do not involve the novel parts of Fig. 3 to do with locally scoped names. Thus this failure of full abstraction has more to do with the nature of continuation-passing transformations than with locally scoped names. Can the CPS transformation we have studied here be modified to give a translation of dynamic allocation into a calculus with Odersky-style local names that is fully abstract? One possibility is to change to a version of $\lambda\nu$ with linear function types $(\multimap)$ and make use of *linearly used continuations*, $((-) \to R) \multimap R$. In particular, it would be interesting to consider the relationship between dynamic allocation and Odersky-style locally scoped names within the *enriched effect calculus* of Egger *et al*, for which the linearly-used CPS translation has a very strong self-duality property [4]. Another possibility is to add locally scoped names to the polymorphic λ-calculus and use *continuations with polymorphic result type*, $\forall R.\,((-) \to R) \to R$, for which the work of Ahmed and Blume [1] suggests there may be a full abstraction result.

Should one care about the full abstraction property? The ν-calculus and the $\lambda\nu$-calculus are not ends in themselves; they are merely vehicles for studying the semantics of higher-order functions with locally scoped names in as simple a setting as possible. One should certainly consider extending the results of this paper to richer languages, beginning by making them Turing-powerful. This could be done by adding fixpoint recursion for functions. It is reasonable to expect the CPS transformation to extend to a computationally adequate translation of such extended languages; whereas any full abstraction result for a modification of the translation probably would not survive such additions.

8 Translating $\lambda\nu$ to ν

Having given a computationally adequate translation of ν-calculus into $\lambda\nu$-calculus, it is natural to consider such a translation in the reverse direction as well. We sketch one in this section, leaving the details for future work.

The main idea is to translate an Odersky-style locally scoped name (at type T say) into the ν-calculus by dynamically generating a fresh name a and then applying to the translated body a function $a\!\setminus_T \in \mathit{Val}(T \to T)$ that implements the operation $c \mapsto a\!\setminus\! c$ in Fig. 3. This

is defined by recursion on the structure of the type T:

$$\begin{aligned} a\!\searrow_{\mathsf{Name}} &= \lambda x \to \mathsf{if}\ x = a\ \mathsf{then}\ \nu a.\, a\ \mathsf{else}\ x \\ a\!\searrow_{\mathsf{Bool}} &= \lambda x \to x \\ a\!\searrow_{T_1 \times T_2} &= \lambda x \to \mathsf{let}\ (x_1\ ,\ x_2) = x\ \mathsf{in}\ (a\!\searrow_{T_1} x_1\ ,\ a\!\searrow_{T_2} x_2) \\ a\!\searrow_{T_1 \to T_2} &= \lambda f \to \lambda x \to a\!\searrow_{T_2} (f\ x). \end{aligned}$$

We would like any computationally adequate translation of $\lambda\nu$-calculus into ν-calculus to be robust with respect to adding extra features such as fixpoint recursion, where the difference between call-by-name and call-by-value becomes visible up to observational equivalence. For a translation to adequately reflect the call-by-name evaluation relation in Fig. 3, one could combine the above idea for implementing Odersky-style $\nu a.\,(-)$ with a standard translation of call-by-name into call-by-value based on using a lifting monad $L(-) = \mathsf{Unit} \to (-)$ to delay evaluation at appropriate points. We did not include a one-element type Unit in the λ-calculus of Sect. 2, but could easily have done so; one could instead use $\mathsf{Bool} \to (-)$ for L. A simpler alternative would be to switch to a call-by-value version of the $\lambda\nu$-calculus.

9 Conclusion

We have shown that Odersky's semantics for $\nu a.\, t$ provides a direct meaning for locally scoped names in higher-order functions (and pairs) from which the more common semantics in terms of dynamic allocation can be recovered via a continuation-passing transformation. This does not help much with understanding the subtle properties of observational equivalence in the Pitts-Stark ν-calculus, because of the complicated nature of the CPS translation. However, the result does shed new light upon the expressive power of the relatively unfamiliar semantics of local names given by Odersky. We have seen that dynamically allocated local names can be encoded with Odersky-style local names. That suggests re-evaluating the usefulness of Odersky's notion. We conclude by mentioning some avenues for doing that.

- Figures 2 and 3 can easily be augmented with evaluation rules for expressions of recursively defined and polymorphic types. We believe our main result (Theorem 5.1) will scale to this extension, using the technique of *step-indexing* to overcome the difficulty of defining a suitable logical relation in the presence of recursive types (see [3], for example). So extended, the $\lambda\nu$-calculus gives a core non-strict functional programming language with Odersky-style local names. For reasons of efficiency one would prefer call-by-need rather than the call-by-name operational semantics in Figure 3. Can one of the standard operational descriptions of call-by-need [9, 17] be combined with this form of locally scoped name? The difficulty is to reconcile its characteristic property of 'scope intrusion', that is, moving $\nu a.\,(-)$ inward to evaluation sites, with local (recursively defined) heaps, $\mathsf{let}\{x_1 = e_1, \ldots, x_n = e_n\}\ \mathsf{in}\ (-)$.
- Odersky's $\nu a.\,(-)$ gives a version of locally scoped names whose evaluation is free of side-effects. Therefore it makes sense to add it to meta-languages for describing the denotational semantics of effects, such as Moggi's computational λ-calculus [11] or the enriched effect calculus of Egger *et al* [4]. Is this a useful extension of such languages?
- Following [14], it should be possible to produce a version of FreshML [19] in which the use of dynamically allocated names is replaced by Odersky-style local names and yet the language still respects α-renaming of bound names in data, up to observational equivalence. The convenient expressive power of FreshML would not be affected and one would regain programming laws for observational equivalence (such as extensionality of

function expressions) that are disrupted by dynamic allocation. (This purer version of FreshML would not pass all of Pottier's criteria for purity [16], since it would admit the anonymous name $\nu a.\, a$ as a value.)

- Fernández and Gabbay [6] consider rewriting for nominal terms extended with 'name generation', a non-binding scoping construct. The formal relationship between this notion and dynamically allocated, or Odersky-style, local names needs clarifying. In any case, the combination of Odersky-style local names with term-rewriting seems worth investigating.

Acknowledgement

Some of the results in this paper are drawn from Lösch's MPhil dissertation; he gratefully acknowledges the support of a Gates Cambridge Scholarship.

References

1 A. Ahmed and M. Blume. Typed closure conversion preserves observational equivalence. In *Proc. ICFP 2008*, pages 157–168. ACM Press.

2 N. Benton and V. Koutavas. A mechanized bisimulation for the nu-calculus. Technical Report MSR-TR-2008-129, Microsoft Research Cambridge, October 2007.

3 D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *Proc. ICFP 2010*, pages 143–156. ACM Press.

4 J. Egger, R. Møgelberg, and A. Simpson. Linearly-used continuations in the enriched effect calculus. In *Proc. FOSSACS 2010*, volume 6014 of Springer *Lecture Notes in Computer Science*, pages 18–32.

5 M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103:235–271, 1992.

6 M. Fernández and M. J. Gabbay. Nominal rewriting with name generation: Abstraction vs. locality. In *Proc. PPDP 2005*, pages 47–58. ACM Press.

7 M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002.

8 S. B. Lassen. Relational reasoning about contexts. In A. D. Gordon and A. M. Pitts, editors, *Higher Order Operational Techniques in Semantics*, pages 91–135. Cambridge University Press, 1998.

9 J. Launchbury. A natural semantics for lazy evaluation. In *Proc. POPL 1993*, pages 144–154. ACM.

10 R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.

11 E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

12 M. Odersky. A functional theory of local names. In *Proc. POPL 1994*, pages 48–59. ACM Press.

13 A. M. Pitts. Typed operational reasoning. In B. C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 7, pages 245–289. The MIT Press, 2005.

14 A. M. Pitts. Structural recursion with locally scoped names. *Journal of Functional Programming*, 21(3):235–286, 2011.

15 A. M. Pitts and I. D. B. Stark. Observable properties of higher order functions that dynamically create local names, or: What's new? In *Proc. MFCS 1993*, volume 711 of Springer *Lecture Notes in Computer Science*, pages 122–141.

16 F. Pottier. Static name control for FreshML. In *Proc. LICS 2007*, pages 356–365. IEEE Computer Society Press.

17 P. Sestoft. Deriving a lazy abstract machine. *Journal of Functional Programming*, 7:231–264, 1997.

18 M. R. Shinwell and A. M. Pitts. On a monadic semantics for freshness. *Theoretical Computer Science*, 342:28–55, 2005.

19 M. R. Shinwell, A. M. Pitts, and M. J. Gabbay. FreshML: Programming with binders made simple. In *Proc. ICFP 2003*, pages 263–274. ACM Press.

20 I. D. B. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, December 1994.

Synthesis from Probabilistic Components*

Yoad Lustig, Sumit Nain, and Moshe Y. Vardi

Department of Computer Science
Rice University, Houston, TX 77005, USA
yoad.lustig@gmail.com , nain@cs.rice.edu , vardi@cs.rice.edu

Abstract

Synthesis is the automatic construction of a system from its specification. In classical synthesis algorithms, it is always assumed that the system is "constructed from scratch" rather than composed from reusable components. This, of course, rarely happens in real life, where almost every non-trivial commercial software system relies heavily on using libraries of reusable components. Furthermore, other contexts, such as web-service orchestration, can be modeled as synthesis of a system from a library of components. Recently, Lustig and Vardi introduced *dataflow* and *control-flow* synthesis from libraries of reusable components. They proved that dataflow synthesis is undecidable, while control-flow synthesis is decidable. In this work, we consider the problem of control-flow synthesis from libraries of *probabilistic components*. We show that this more general problem is also decidable.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Temporal synthesis, probabilistic components

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.412

1 Introduction

Hardware and software systems are rarely built from scratch. Almost every non-trivial system is based on existing components. A typical component might be used in the design of multiple systems. Examples of such components include function libraries, web APIs, and ASICs. Consider the mapping application in a typical smartphone. Such an application might call the location service provided by the phone's operating system to get the user's co-ordinates, then call a web API to obtain the correct map image tiles, and finally call a graphics library to display the user's location on the screen. None of these components are exclusive to the mapping application and all of them are commonly used by other applications.

The construction of systems from reusable components is an area of active research. Some examples of important work on the subject can be found in Sifakis' work on component-based construction [15], and de Alfaro and Henzinger's work on "interface-based design" [7]. Furthermore, other situations, such as web-service orchestration [2], can be viewed as the construction of systems from libraries of reusable components.

Synthesis is the automated construction of a system from its specification. In contrast to model checking, which involves verifying that a system satisfies the given specification, synthesis aims to automatically construct the required system from its formal specification. The modern approach to temporal synthesis was initiated by Pnueli and Rosner who introduced linear temporal logic (LTL) synthesis [13]. In LTL synthesis, the specification is given in LTL and the system constructed is a finite-state transducer modeling a reactive

* Work supported in part by NSF grants CCF-0728882, and CNS 1049862, by BSF grant 9800096, and by gift from Intel.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 412–427

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

system. In this setting it is always assumed that the system is "constructed from scratch" rather than "composed" from existing components. Recently, Lustig and Vardi [11] introduced the study of synthesis from reusable components. The use of components abstracts much of the detailed behavior of a sub-system, and allows one to write specifications that mention only the aspects of sub-systems relevant for the synthesis of the system at large.

A major concern in the study of synthesis from reusable components is the choice of a mathematical model for the components and their composition. The exact nature of the reusable components in a software library may differ. One finds in the literature many different types of components; for example, function libraries (for procedural programming languages) or object libraries (for object-oriented programming languages). Indeed, there is no single "right" model encompassing all possible facets of the problem. The problem of synthesis from reusable components is a general problem to which there are as many facets as there are models for components and types of composition [15].

As a basic model for a component, following [11], we abstract away the precise details of the component and model a component as a *transducer*, i.e., a finite-state machine with outputs. Transducers constitute a canonical model for reactive components, abstracting away internal architecture and focusing on modeling input/output behavior. In [11], two models of composition were studied. In *dataflow* composition, the output of one component is fed as input to another component. The synthesis problem for dataflow composition was shown to be undecidable. In *control-flow* composition control is held by a single component at every point in time. The synthesis problem can then be viewed as constructing a supervisory transducer that switches control between the component transducers. Control-flow composition is motivated by software (and web services) in which a single function is in control at every point during the execution. LTL synthesis in this setting was shown in [11] to be 2EXPTIME-complete, just like classical LTL synthesis [13].

In this paper, we extend the control-flow synthesis model of [11] to probabilistic components, which are transducers with a probabilistic transition function. This is a well known approach to modeling systems where there is probabilistic uncertainty about the results of input actions. Intuitively, we aim at constructing a reliable system from unreliable components. There is a rich literature about verification and analysis of such systems, cf. [16, 5, 6, 17], as well about synthesis in the face of probabilistic uncertainty [1]. The introduction of probability requires us to use a probabilistic notion of correctness; here we choose the *qualitative* criterion that the specification be satisfied with probability 1, leaving the study of *quantitative criteria* to future work.

Here, our focus is on proving decidability, rather than on establishing precise complexity bounds, leaving the study of precise bounds to future work. Consequently, we abstract away from the details of the specification formalism and assume that the specification is given in terms of deterministic parity word automata (DPW). This allows us to consider all ω-regular properties. We define and study the *DPW probabilistic realizability* and synthesis problems, where the input is a library $\mathcal{L}$ of probabilistic components and a DPW $\mathcal{A}$, and the question is whether one can construct a *finite* system S from the components in $\mathcal{L}$, such that, regardless of the external environment, the traces generated by the system S are accepted by $\mathcal{A}$ with probability 1. Each component in the library can be used an arbitrary number of times in the construction and there is no apriori bound on the size of the system obtained. The technical challenge here is dealing with the finiteness of the system under construction. In [11], as well as in [13], one need not deal with finiteness from the start. In fact, one can test realizability without being concerned with finiteness of the constructed system, as finiteness is a *consequence* of the construction. This is not the case here, where we need to

deal with finiteness from the start. Nevertheless, we are able to show that the problem is in 2EXPTIME.

Before tackling the full problem, we first consider a restricted version of the problem, where the specification is given in the form of a parity index on the states of the components, and the composed system must satisfy the parity condition. We call this the *embedded parity realizability* problem. We solve this problem and then show how solving the embedded parity realizability problem directly allows us to solve the more general DPW probabilistic realizability problem as well. The key idea here is that by taking the product of the specification DPW with each of the components, we can obtain larger components each of whose states has a parity associated with it. The challenge in completing the reduction is the need to generate a static composition, which does not depend on the history of the computation. Here we use ideas about synthesis with incomplete information from [10].

The paper is self-contained, except for certain technical proofs that have been omitted to save space; a longer version is posted on the authors' home pages.

2 Preliminaries

Given a set D of directions, a *D-tree* is a set $T \subseteq D^*$ such that if $x \cdot c \in T$, where $x \in D^*$ and $c \in D$, then also $x \in T$. For every $x \in T$, the words $x \cdot c$, for $c \in D$, are the *successors* of x. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$, either x is a leaf or there exists a unique $c \in D$ such that $x \cdot c \in \pi$. The *full* D-tree is D^*. Given an alphabet Σ, a *Σ-labeled D-tree* is a pair $\langle T, \tau \rangle$, where T is a tree and $\tau : T \to \Sigma$ maps each node of T to a letter in Σ. A *subtree* of $\langle D^*, \tau \rangle$, is a Σ-labeled D-tree $\langle T, \tau' \rangle$, where $\tau'(x) = \tau(x)$, for all $x \in T$. For a node $x \in D^*$, the *full subtree* at x is the subtree whose set of nodes is $x \cdot D^*$.

A *deterministic transducer* is a tuple $B = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, L \rangle$, where: Σ_I is a finite input alphabet, Σ_O is a finite output alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $L : Q \to \Sigma_O$ is an output function labeling states with output letters, and $\delta : Q \times \Sigma_I \to Q$ is a transition function. We define $\delta^* : \Sigma_I^* \to Q$ as follows: $\delta^*(\epsilon) = q_0$ and for $x \in \Sigma_I^*$ and $a \in \Sigma_I$, $\delta^*(x \cdot a) = \delta(\delta^*(x), a)$. We denote by $tree(B)$, the Σ_O-labeled Σ_I-tree $\langle \Sigma_I^*, \tau \rangle$, where for all $x \in \Sigma_I^*$, we have $\tau(x) = L(\delta^*(x))$. We say $tree(B)$ is the *unwinding* of B. A Σ-labeled D-tree T is called *regular*, if there exists a deterministic transducer C such that $T = tree(C)$.

Given a directed graph $G = (V, E)$, a *strongly connected component* of G is a subset U of V, such that for all $u, v \in U$, u is reachable from v. We can define a natural partial order on the set of maximal strongly connected components of G as follows: $U_1 \leq U_2$ if there exists $u_1 \in U_1$ and $u_2 \in U_2$ such that u_1 is reachable from u_2. Then $U \subseteq V$ is an *ergodic set* of G if it is a minimal element of the partial order.

A probability distribution on a finite set X is a function $f : X \to [0, 1]$ such that $\sum_{x \in X} f(x) = 1$. We use $Dist(X)$ to denote the set of all probability distributions on set X. A *probabilistic transducer*, is a tuple $\mathcal{T} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, F, L \rangle$, where: Σ_I is a finite input alphabet, Σ_O is a finite output alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : (Q - F) \times \Sigma_I \to Dist(Q)$ is a probabilistic transition function, $F \subseteq Q$ is a set of exit states, and $L : Q \to \Sigma_O$ is an output function labeling states with output letters. Note that there are no transitions out of an exit state. If F is empty, we say $\mathcal{T}$ is a probabilistic transducer without exits.

Given a probabilistic transducer $M = (\Sigma_I, \Sigma_o, Q, q_0, \delta, F, L)$, a *strategy* for M is a function $f : Q^* \to Dist(\Sigma_I)$ that probabilistically chooses an input for each sequence of states. A strategy is memoryless if the choice depends only on the last state in the sequence. A memoryless strategy can be written as a function $g : Q \to Dist(\Sigma_I)$. A strategy is *pure* if

the choice is deterministic. A pure strategy is a function $h : Q^* \to \Sigma_I$, and a memoryless and pure strategy is a function $h : Q \to \Sigma_I$.

A strategy f along with a probabilistic transducer M, with set of states Q, induces a probability distribution on Q^ω, denoted μ_f. By standard measure theoretic arguments, it suffices to define μ_f for the cylinders of Q^ω, which are sets of the form $\beta \cdot Q^\omega$, where $\beta \in Q^*$. First we extend δ to exit states as follows: for $a \in \Sigma_I$, $q \in F$, $q' \in Q$, $\delta(q, a)(q) = 1$ and $\delta(q, a)(q') = 0$ when $q' \neq q$. Then we define $\mu_f(q_0 \cdot Q^\omega) = 1$, and for $\beta \in Q^*$, $q, q' \in Q$, $\mu_f(\beta q q' \cdot Q^\omega) = \mu_f(\beta q)(\sum_{a \in \Sigma_I} f(\beta q)(a) \times \delta(q, a)(q'))$. These conditions say that there is a unique start state, and the probability of visiting a state q', after visiting βq, is the same as the probability of the strategy picking a particular letter multiplied by the probability that the transducer transitions from q to q' on that input letter, summed over all input letters.

Let M be a probabilistic transducer, Q be its set of states, and f be a memoryless strategy for M. We define the graph induced by f on Q, denoted by $G_{M,f}$, as the directed graph (Q, E), where $(q_1, q_2) \in E$ if $\sum_{a \in \Sigma_I} f(q_1)(a)\, \delta(q_1, a)(q_2) > 0$. That is, there is an edge from q_1 to q_2 if the transducer can transition from the state q_1 to the state q_2 on an input letter that the strategy chooses with positive probability. Given $q_1, q_2 \in Q$, we say that q_2 is reachable from q_1 if there is a path from q_1 to q_2 in $G_{M,f}$. We say a state is ergodic if it belongs to some ergodic set of $G_{M,f}$. An ergodic set is reachable if there is a path from the start state to some state in the ergodic set. A state q of M is *reachable under* f, if there is a path in $G_{M,f}$ from q_0 to q.

A *library* is a set of probabilistic transducers that share the same input and output alphabets. Each transducer in the library is called a *component*. Given a finite set of directions D, we say a library $\mathcal{L}$ has width D, if each component in the library has exactly $|D|$ exit states. Since we can always add dummy unreachable exit states to any component, we assume, w.l.o.g., that all libraries have an associated width, usually denoted D. In the context of a particular component, we often refer to elements of D as exits, and subsets of D as sets of exits. Given a component M from library $\mathcal{L}$, and a strategy f for M, we say that the exit $i \in D$ is *selected* by f, if the ith exit state of M is reachable under f.

An *index function* for a transducer is a function that assigns a natural number, called a priority index, to each state of the transducer. An index function for a library is a function that assigns a priority to every state of every component in the library. Given an index function α for a library $\mathcal{L}$, we define $\max(\alpha)$ to be the highest priority assigned by α. We can assume, w.l.o.g., that $\max(\alpha)$ is not larger than twice the maximal number of states in the components of the library. Given a transducer M, index function α, and a strategy f for M, we say f *visits* priority p if there exists a state q of M such that $\alpha(q) = p$ and q is reachable under f.

3 Control-flow Composition from Libraries

We first informally describe our notion of control-flow composition of components from a library. The components in the composition take turns interacting with the environment, and at each point in time, exactly one component is active. When the active component reaches an exit state, control is transferred to some other component. Thus, to define a control flow composition, it suffices to name the components used and describe how control should be transferred between them. We use a deterministic transducer to define the transfer of control. Each library component can be used multiple times in a composition, and we treat these occurrences as distinct *component instances*. We emphasize that the composition can contain potentially arbitrarily many repetitions of each component inside it. Thus, the

size of the composition, a priori, is not bounded. Note that our notion of composition is *static*, where the components called are determined before run time, rather than *dynamic*, where the components called are determined during run time.

Let $\mathcal{L}$ be a library with width D. A *composer* over $\mathcal{L}$ is a deterministic tranducer $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$. Here $\mathcal{M}$ is an arbitrary finite set of states. There is no bound on the size of $\mathcal{M}$. Each $\mathsf{M}_i \in \mathcal{M}$ is the name of an instance of a component from $\mathcal{L}$ and $\lambda(\mathsf{M}_i) \in \mathcal{L}$ is the type of M_i. We use the following notational convention for component instances and names: the upright letter M always denotes component names (i.e. states of a composer) and the italicized letter M always denotes the corresponding component instances (i.e. elements of $\mathcal{L}$). Further, for notational convenience we often write M_i directly instead of $\lambda(\mathsf{M}_i)$. Note that while each M_i is distinct, the corresponding components M_i need not be distinct. Each composer defines a unique composition over components from $\mathcal{L}$. The current state of the composer corresponds to the component that is in control. The transition function Δ describes how to transfer control between components: $\Delta(\mathsf{M}, i) = \mathsf{M}'$ denotes that when the composition is in the ith final state of component M it moves to the start state of component M'. A composer can be viewed as an implicit representation of a composition. We give an explicit definition of composition below.

▶ **Definition 1** (Control-flow Composition)**.** Let $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$ be a composer over library $\mathcal{L}$ with width D, such that $\mathcal{M} = \{\mathsf{M}_0, \ldots, \mathsf{M}_n\}$, $\lambda(\mathsf{M}_i) = (\Sigma_I, \Sigma_O, Q_i, q_0^i, \delta_i, F_i, L_i)$ and $F_i = \{q_x^i : x \in D\}$. The composition defined by C, denoted $\mathcal{T}_C$, is a probabilistic transducer $\langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \emptyset, L\rangle$, where $Q = \bigcup_{i=0}^{n}(Q_i \times \{i\})$, $q_0 = \langle q_0^0, 0\rangle$, $L(\langle q, i\rangle) = L_i(q)$, and the transition function δ is defined as follows: For $\sigma \in \Sigma_I$, $\langle q, i\rangle \in Q$ and $\langle q', j\rangle \in Q$,

1. If $q \in Q_i \setminus F_i$, then

$$\delta(\langle q, i\rangle, \sigma)(\langle q', j\rangle) = \begin{cases} \delta_i(q, \sigma)(q') & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

2. If $q = q_x^i \in F_i$, where $\Delta(\mathsf{M}_i, x) = \mathsf{M}_k$, then

$$\delta(\langle q, i\rangle, \sigma)(\langle q', j\rangle) = \begin{cases} 1 & \text{if } j = k \text{ and } q' = q_0^k \\ 0 & \text{otherwise} \end{cases}$$

Note that the composition is a probabilistic transducer without exits. When the composition is in a state $\langle q, i\rangle$ corresponding to a non-exit state q of component M_i, it behaves like M_i. When the composition is in a state $\langle q_f, i\rangle$ corresponding to an exit state q_f of component M_i, the control is transferred to the start state of another component as determined by the transition function of the composer. Thus, at each point in time, only one component is active and interacting with the environment.

4 Synthesis for Embedded Parity

In this section we consider a simplified version of the general synthesis problem, where each state of a component in the library has a priority associated with it and the specification to be satisfied is that the highest priority visited i.o. must be even with probability 1.

Let M be a probabilistic tranducer and α be an index function. A strategy f for M is *winning* for the environment if with positive probability the highest priority visited infinitely often (i.o.) is odd. We say that M *satisfies* α if there exists no winning strategy for the environment. Given a composer C over library $\mathcal{L}$, we say that C *satisfies* α if $\mathcal{T}_C$ satisfies α.

Given a library $\mathcal{L}$ with width D, an *exit control relation* is a set $R \subseteq D \times \mathcal{L}$. We say that a composer $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$ over $\mathcal{L}$ is *compatible* with R, if the following holds: for all $\mathsf{M}, \mathsf{M}' \in \mathcal{M}$ and $i \in D$, if $\Delta(\mathsf{M}, i) = \mathsf{M}'$ then $(i, M') \in R$. Thus, each element of R can be viewed as a constraint on how the composer is allowed to connect components.

▶ **Definition 2.** The *embedded parity realizability problem* is: Given a library $\mathcal{L}$ with width D, an exit control relation R for $\mathcal{L}$, and an index function α for $\mathcal{L}$, decide whether there exists a composer C over $\mathcal{L}$, such that C satisfies α and C is compatible with R. If such a composer exists, we say that $\mathcal{L}$ *realizes* α under R. The *embedded parity synthesis problem* is to find such a composer C if it exists.

The following theorem allows us to restrict attention to memoryless strategies. It states that if a winning strategy exists, then a memoryless winning strategy must also exist. Here we give a direct combinatorial proof, but we note that the result can also be obtained by adapting the methods in [4], where a similar result was proved for 2–1/2 player stochastic parity games by Chatterjee et al.

▶ **Theorem 3.** *Given a probabilistic transducer M, and index function α, if there exists a winning strategy for the environment then there exists a pure and memoryless winning strategy.*

Memoryless strategies are important because they induce an ergodic structure on the set of states. Ergodic sets are useful because they enable us to replace probabilistic reasoning with combinatorial reasoning. In particular, they have the following crucial properties: (a) the suffix of a path is contained in some ergodic set with probability 1, and (b) the suffix of a path is contained in a proper subset of an ergodic set with probability zero [9]. This allows us to define the winning strategy condition in terms of graph reachability.

▶ **Lemma 4.** *Let M be a probabilistic transducer and f be a memoryless strategy for M. Then f is winning for the environment iff $G_{M,f}$ has a reachable ergodic set whose highest priority is odd.*

When the underlying probabilistic transducer is a composition, ergodic sets acquire additional structure. Given a composer C and a memoryless strategy f for $\mathcal{T}_C$, if a reachable ergodic set X of $G_{\mathcal{T}_C,f}$ contains some state from a component M of $\mathcal{T}_C$, then either X is contained in M or all the reachable states of M are contained in X. Formally:

▶ **Lemma 5.** *Let $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$ be a composer over $\mathcal{L}$ and f be a memoryless strategy for $\mathcal{T}_C$. Let $\mathsf{M}_i \in \mathcal{M}$ and Q_i be the state space of M_i. Let X be a reachable ergodic set of $G_{\mathcal{T}_C,f}$ such that $X \cap (Q_i \times \{i\}) \neq \emptyset$. Then either $X \subseteq Q_i \times \{i\}$ or $(Q_i \times \{i\}) \cap Y \subseteq X$, where Y is the set of states of $\mathcal{T}_C$ that are reachable under f.*

Proof. Assume that $X \cap (Q_i \times \{i\}) \neq \emptyset$ and X is not contained in $Q_i \times \{i\}$. Let $(q, i) \in X \cap (Q_i \times \{i\})$ and $(q', j) \in X - (Q_i \times \{i\})$, for some $j \neq i$. Since X is ergodic, there is a path π in $G_{\mathcal{T}_C,f}$ from (q', j) to (q, i). Let s be the first state along π such that $s = (q'', i) \in Q_i \times \{i\}$. We claim that $q'' = q_0^i$, where q_0^i is the start state of M_i. Let $s' = (q''', k)$, where $k \neq i$, be the predecessor of s in π. By the definition of $G_{\mathcal{T}_C,f}$, there is an edge from s' to s only if $\mathcal{T}_C$ can transition from s' to s on some input with positive probability. By Definition 1, $\mathcal{T}_C$ can transition from (q''', k) to (q'', i) only if q''' is a final state of M_k and q'' is the initial state of M_i. Thus (q_0^i, i) is in X.

Since X is an ergodic set, if it contains a state s of $\mathcal{T}_C$, then it also contains all states reachable under f from s. By definition, every state in $(Q_i \times \{i\}) \cap Y$ is reachable under f from (q_0^i, i). Since X contains (q_0^i, i), it also contains all states in $(Q_i \times \{i\}) \cap Y$. ◀

Given a graph G, each of whose vertices is assigned a priority, we say that G has the *odd ergodic property* if it has a reachable ergodic set whose highest priority is odd. Consider a composer C and a memoryless strategy f for $\mathcal{T}_C$. Then, by Lemma 4, f is winning for the environment iff $G_{\mathcal{T}_C,f}$ has the odd ergodic property. So the probabilistic notion of winning strategy is reduced to a combinatorial one. However, the graph $G_{\mathcal{T}_C,f}$ is very large as it contains all the internal states of each component explicitly. Further, to show that C satisfies α, we have to consider every possible memoryless strategy for C. We tackle this complexity by simplifying the description of a strategy f and graph $G_{\mathcal{T}_C,f}$ so as to abstract away the inner states of components and the choices that f makes on those inner states. Let $\mathcal{M}$ be the state space of C. We aim to replace $G_{\mathcal{T}_C,f}$ by a simpler graph G', whose set of vertices is $\mathcal{M}$, such that the odd ergodic property is preserved. We first discuss this transformation informally, and then give formal definitions and proofs.

Let M be a component of $\mathcal{T}_C$. If some reachable ergodic set of $G_{\mathcal{T}_C,f}$ lies entirely within M, we say M is a *sink*. When the highest priority in the ergodic set is odd (resp. even) we say M is an *odd* (resp. *even*) sink for f. Note that a component can be both an odd and an even sink for a given strategy. Intuitively, we aim to replace the subgraph of $G_{\mathcal{T}_C,f}$ that corresponds to states of M by a single new vertex x_M to obtain a new graph G' and assign a suitable priority to x_M such that the odd ergodic property is preserved by the transformation. Now if M is not a sink, then, by Lemma 5, x_M lies in a reachable ergodic set of G' iff all reachable states of M lie in a reachable ergodic set of $G_{\mathcal{T}_C,f}$. In this case, we can simply assign the highest reachable priority in M to x_M and the odd ergodic property is preserved. If, however, M is a sink, then the collapse of M to a single vertex might introduce new ergodic sets in the graph. That is, x_M might lie in an ergodic set of G' which has no analogue in $G_{\mathcal{T}_C,f}$. We then have to choose the priority of x_M such that the odd ergodic property is still preserved. There are two cases to consider:

- M is an odd sink for f. Then, by Lemma 4, f is winning for the environment. Let f_M denote f restricted to the states in M. Then f_M is a memoryless strategy for M that is winning for the environment, and in every composition involving M, the environment can simply play f_M on the states in M to win. So a component that is an odd sink is not useful for synthesizing compositions. We note that it is easy to check for and remove any odd sinks from $\mathcal{L}$ in a preprocessing step before attempting synthesis. Checking whether a particular component is a sink is equivalent to model checking Markov decision processes and can be done in polynomial time [16]. In the rest of the paper, we assume that the given library $\mathcal{L}$ does not contain components that are odd sinks.
- M is an even sink for f but not an odd sink for f. Then, by Lemma 5, every reachable state in M either lies in an even sink or does not lie in an ergodic set. So no reachable state in M is part of an ergodic set with odd highest priority. Thus collapsing M to x_M does not remove any ergodic sets with odd highest priority. It only remains to consider the possibility that the transformation can introduce a new ergodic set whose highest priority is odd. We can avoid this by assigning a priority of $2\max(\alpha)$ to x_M, where $\max(\alpha)$ is the highest parity assigned by the index function α. Then if x_M is part of a reachable ergodic set X' in G', then X' has highest priority $2\max(\alpha)$, which is even. Thus the odd ergodic property is preserved.

In formalizing the approach given above, instead of explicitly transforming $G_{\mathcal{T}_C,f}$ into a more abstract graph, it is simpler to directly define a suitable graph on the state space $\mathcal{M}$ of the composer C such that the odd ergodic property is preserved. Just as a memoryless strategy f applied to the composition $\mathcal{T}_C$ gives rise to the graph $G_{\mathcal{T}_C,f}$, we define a combinatorial

object, called a *choice function*, such that choice function g together with composer C gives rise to a graph $G_{C,g}$.

▶ **Definition 6** (Choice Function)**.** Given a library $\mathcal{L}$ with width D and index function α, we define the set $LABELS(\mathcal{L}) \subseteq 2^D \times \{1,\ldots,2\max(\alpha)\} \times \mathcal{L}$ as follows: $(X,j,M) \in LABELS(\mathcal{L})$ iff there exists a memoryless strategy f for M such that

- $X \subseteq D$ is the set of exits of selected by f in M.
- If M is an even sink for f, then $j = 2\max(\alpha)$.
- Otherwise j is the highest priority visited by f in M.

Given a composer $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$ over $\mathcal{L}$, a *choice function* for C, is a function $g : \mathcal{M} \to 2^D \times \{1,\ldots,2\max(\alpha)\}$, such that, for all $\mathsf{M}_i \in \mathcal{M}$, $(g(\mathsf{M}_i), M_i) \in LABELS(\mathcal{L})$. The graph induced by g on C, denoted $G_{C,g}$, is the directed graph $(\mathcal{M}, \mathcal{E})$, where $(\mathsf{M}_1, \mathsf{M}_2) \in \mathcal{E}$ if $\Delta(\mathsf{M}_1, i) = \mathsf{M}_2$ for some $i \in D$ such that $i \in X$ where $g(\mathsf{M}_1) = (X, j)$. The priority of a vertex $\mathsf{M} \in \mathcal{M}$ of $G_{C,g}$ is j where $g(\mathsf{M}) = (X, j)$. We say that g has *rank* r, if $G_{C,g}$ has a reachable ergodic set whose highest priority is r.

The size of the set $LABELS(\mathcal{L})$ is at most $\max(\alpha)|\mathcal{L}|2^{|D|}$. For an arbitrary triple (X,j,M), we can check whether $(X,j,M) \in LABELS(\mathcal{L})$ in time polynomial in $|M|$ using standard techniques for solving Markov decision processes [16]. Thus $LABELS(\mathcal{L})$ can be computed in time exponential in the size of $\mathcal{L}$.

▶ **Theorem 7.** *Let C be a composer over $\mathcal{L}$. Then there exists a strategy for $\mathcal{T}_C$ that is winning for the environment iff there exists a choice function for C that has an odd rank.*

Proof. Let $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$. Let Q_i be the state space of $M_i = \lambda(\mathsf{M}_i)$, for $\mathsf{M}_i \in \mathcal{M}$, and let $Q = \bigcup(Q_i \times \{i\})$ be the state space of $\mathcal{T}_C$.

Only If: Assume there exists a strategy for $\mathcal{T}_C$ that is winning for the environment. Then, by Theorem 3, there exists a memoryless winning strategy f. We construct a choice function g for C as follows: for all $\mathsf{M}_i \in \mathcal{M}$, $g(\mathsf{M}_i) = (X, p)$, where X is the set of exits of M_i selected by f, and $p = 2\max(\alpha)$ if M_i is an even sink for f and otherwise p is the highest priority in M_i visited by f. Since f is winning, $G_{\mathcal{T}_C,f}$ has a reachable ergodic set H with odd highest priority r. Consider the set $\mathcal{H} \subseteq \mathcal{M}$ defined as follows: for all $\mathsf{M}_i \in \mathcal{M}$, $\mathsf{M}_i \in \mathcal{H}$ if $(Q_i \times \{i\}) \cap H \neq \emptyset$. Thus, $\mathcal{H}$ contains a state of the composer C if the corresponding component of $\mathcal{T}_C$ overlaps with the ergodic set H. Since $\mathcal{L}$ contains no components that are odd sinks, and even sinks can not be a part of an ergodic set whose highest priority is odd, H must contain all the reachable states in each component named in $\mathcal{H}$.

We claim that $\mathcal{H}$ is an ergodic set of $G_{C,g}$. We first show that $\mathcal{H}$ is strongly connected. Let M_i and M_k be in $\mathcal{H}$. Since all the reachable states of M_i and M_k are contained in H, in particular their start states are also contained in H. Let these be q_i and q_k respectively. Then there is a path in $G_{\mathcal{T}_C,f}$ from (q_i, i) to (q_k, k) because H is an ergodic set of $G_{\mathcal{T}_C,f}$. Consider the path π from (q_i, i) to (q_k, k) that contains the least number of exit states. Let the length of π be n and let (q'_i, i) be the first exit state along π. Suppose $\Delta(\mathsf{M}_i, x) = \mathsf{M}_j$, where q'_i is the exit state of M_i in direction x, and let q_j be the start state of M_j. Then, if $g(\mathsf{M}_i) = (X, p)$, we have $x \in X$, so there is an edge from M_i to M_j in $G_{C,g}$, and the immediate next state after (q'_i, i) in π is (q_j, j). The suffix of π starting from (q_j, j) is a path π' from (q_j, j) to (q_k, k) of length less than n. Further, by construction, among all such paths it has the least number of exit states. Assume, by the induction hypothesis, there is a path from M_j to M_k in $G_{C,g}$. Since $(\mathsf{M}_i, \mathsf{M}_j)$ is also an edge in $G_{C,g}$, therefore, by induction,

there is a path from M_i to M_k in $G_{C,g}$. M_i and M_k were chosen arbitrarily in $\mathcal{H}$. So $\mathcal{H}$ is strongly connected.

Next, we show that there are no edges that leave $\mathcal{H}$. Assume there is some edge in $G_{C,g}$ from a vertex $\mathsf{M}_i \in \mathcal{H}$ to a vertex $\mathsf{M}_j \in \mathcal{M} - \mathcal{H}$. Let $g(\mathsf{M}_i) = (X, p')$. Then there exists $x \in X$ such that $\Delta(\mathsf{M}_i, x) = \mathsf{M}_j$. Let (q', i) be the exit state of M_i in direction x. Then (q', i) is reachable under f and so is (q_j, j), where q_j is the start state of M_j. Therefore, there is an edge in $G_{\mathcal{T}_C,f}$ from $(q', i) \in H$ to $(q_j, j) \notin H$, which contradicts that H is an ergodic set. Thus no edges leave $\mathcal{H}$ in $G_{C,g}$ and $\mathcal{H}$ is ergodic.

Finally, we show that the highest priority in $\mathcal{H}$ is r. By construction of g, since H does not contain any even sinks, the priority of a vertex M_i in $\mathcal{H}$ is the highest priority visited in M_i by f. Thus, the highest priority in $\mathcal{H}$ is at most the highest priority in H, which is r. Let $(q, j) \in H$ be such that q has priority r. Then the highest priority visited by f in M_j is r, so $g(\mathsf{M}_j) = (X, r)$ for some $X \subseteq D$. Since $\mathsf{M}_j \in \mathcal{H}$, the highest priority in $\mathcal{H}$ is r, and g has rank r.

If: Now assume that g is a choice function for C with rank p, for some odd $p \leq \max(\alpha)$. Then, by the definition of choice function, for all $\mathsf{M}_i \in \mathcal{M}$, there exists a memoryless strategy f_i for M_i, such that $g(\mathsf{M}_i) = (X_i, p_i)$ where X_i is the set of exit directions of M_i under f_i, and $p_i = 2\max(\alpha)$ if M_i is an even sink for f_i and otherwise p_i is the highest priority visited by f_i.

We define a memoryless strategy f for $\mathcal{T}_C$ as follows: for all $q \in Q_i$, $f(q, i) = f_i(q)$. Since g has rank p, there exists a reachable ergodic set $\mathcal{H} \subseteq \mathcal{M}$ of $G_{C,g}$ with highest priority p. Consider the set $H = \{(q, i) : q \in Q_i, \mathsf{M}_i \in \mathcal{H}\}$, which consists of all states in all components corresponding to the set $\mathcal{H}$. Let H_f be the subset of H that is reachable under f from the start state of $\mathcal{T}_C$. We first show that H_f is strongly connected. Let (q_i, i) and (q_k, k) be two arbitrary states in H_f. Then q_i is a state of M_i and q_k is a state of M_k. Further, M_i and M_k are both in $\mathcal{H}$. We have the following two cases:

1. *q_i is the start state of M_i.* Consider the shortest path in $G_{C,g}$ from M_i to M_k. Such a path exists because $\mathcal{H}$ is an ergodic set of $G_{C,g}$. Let the length of the path be n and let M_j be the successor of M_i in this path. So there is path of length $n-1$ in $G_{C,g}$ from M_j to M_k. Now, by the definition of $G_{C,g}$, there exists $x \in D$ such that $\Delta(\mathsf{M}_i, x) = \mathsf{M}_j$ and the exit state in direction x is reachable from the start state of M_i under f_i. Thus there is a path in $G_{\mathcal{T}_C,f}$ from (q_i, i) to (q_j, j) where q_j is the start state of M_j. By induction, there is a path in $G_{\mathcal{T}_C,f}$ from (q_i, i) to (q_k, k).
2. *q_i is not the start state of M_i.* Let $g(\mathsf{M}_i) = (X, p')$, where $X \subseteq D$. Since p is the highest priority in $\mathcal{H}$ and $\mathsf{M}_i \in \mathcal{H}$, we have $p' \leq p \leq \max(\alpha)$. Thus $p' \neq 2\max(\alpha)$ and so M_i is not an even sink for f. Also, the library $\mathcal{L}$ is assumed to have no components that are odd sinks. Thus, some exit of M_i must be reachable from q_i under f_i. Let this exit be in direction $x \in D$, and let $\Delta(\mathsf{M}_i, x) = \mathsf{M}_j$. Then there is a path in $G_{\mathcal{T}_C,f}$ from (q_i, i) to (q_j, j) where q_j is the start state of M_j. Now, since q_j is a start state, by the previous case, there is a path from (q_j, j) to (q_k, k) in $G_{\mathcal{T}_C,f}$. So there is a path from (q_i, i) to (q_k, k) and therefore H_f is strongly connected.

Assume that some edge in $G_{\mathcal{T}_C,f}$ leaves H_f. Let there be an edge between $(q, i) \in H_f$ and $(q', j) \in Q - H_f$. Now M_j can not belong to $\mathcal{H}$ because otherwise (q', j) would be in H_f. So we have $i \neq j$ and (q, i) must be an exit state of M_i. Therefore there is an edge in $G_{C,g}$ from $\mathsf{M}_i \in \mathcal{H}$ to $\mathsf{M}_j \in \mathcal{M} - \mathcal{H}$, which contradicts that $\mathcal{H}$ is ergodic. Thus H_f is also an ergodic set.

By Lemma 4, it suffices to show that the highest priority in H_f is odd. Now p is the highest priority in $\mathcal{H}$, and p is odd, which means $p \neq 2\max(\alpha)$. So there must exist $\mathsf{M}_i \in \mathcal{H}$ such that some state q in M_i has priority p and is reachable under f_i. Then (q, i) is in H_f and so H_f has highest priority at least p. Assume some state (q', j) in H_f has priority $p' > p$. Since q' is reachable under f_j, therefore, we have $g(\mathsf{M}_j) = (X, p'')$, for some $X \subseteq D$ and $p'' \geq p' > p$. This contradicts the fact that $\mathsf{M}_j \in \mathcal{H}$. Thus the highest priority in the ergodic set H_f is p, which is odd. ◀

Let $\Gamma = LABELS(\mathcal{L})$. A composer and choice function pair has a natural representation as a regular Γ-labeled D-tree. Given a composer $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$ over $\mathcal{L}$, and a choice function g for C, we denote by $tree(C, g)$, the regular Γ-labeled full D-tree $\langle D^*, \tau\rangle$, where for all $x \in D^*$, we have that $\tau(x) = (g(\Delta^*(x)), \lambda(\Delta^*(x)))$. Thus $tree(C, g)$ is the tree obtained as a result of adding labels to $tree(C)$ such that a node x corresponding to $\mathsf{M}_i \in \mathcal{M}$ that is labeled with M_i in $tree(C)$ is labeled with (X, j, M_i) where $(X, j) = g(\mathsf{M}_i)$. As we show in the next lemma, the mapping is reversible, in the sense that given a regular Γ-labeled D-tree, we can obtain a composer and choice function in a natural way.

▶ **Lemma 8.** *Let T be a regular Γ-labeled full D-tree. Then there exist a composer C over $\mathcal{L}$ and a choice function g for C such that $tree(C, g) = T$.*

In light of Lemma 8, we can represent an arbitrary regular Γ-labeled full D-tree as $tree(C, g)$ for some composer C over $\mathcal{L}$ and some choice function g for C. Similarly, we can represent an arbitrary regular $\mathcal{L}$-labeled full D-tree as $tree(C)$ for some composer C over $\mathcal{L}$.

Since the question of whether a given composition satisfies α boils down to whether its composer has a choice function that has an odd rank, we find it useful to characterize regular trees that correspond to choice functions having a particular rank (see [14] for related results). First, we inductively define the set of *marked* nodes of a Γ-labeled D-tree as follows: the root is always marked, and a node $y \cdot i$, where $i \in D$ and $y \in D^*$, is marked if y is marked and $i \in X$, where (X, j, M) is the label on $y \cdot i$.

▶ **Lemma 9.** *Let $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$ be a composer over library $\mathcal{L}$ with width D, α be an index function for $\mathcal{L}$, g be a choice function for C, and $p \leq \max(\alpha)$. Then g has rank p iff $tree(C, g)$ has a full subtree T such that:*

1. *The root of T is marked.*
2. *Every node in T that is marked has priority label at most p.*
3. *From each marked node in T there is a path in T to a marked node with priority label p.*

The conditions given by Lemma 9 can be checked by a suitable tree automaton as follows:

▶ **Lemma 10.** *Let $\mathcal{L}$ be a library with width D and let $p \leq k$. Then there exists an nondeterministic Büchi tree automaton (NBT) $\mathcal{A}_p$ such that $\mathcal{A}_p$ accepts a Γ-labeled regular D-tree T iff $T = tree(C, g)$ for some composer C over $\mathcal{L}$ and choice function g with rank p.*

Proof. By Lemma 8 and 9, it suffices to construct an NBT $\mathcal{A}_p$ such that $\mathcal{A}_p$ accepts a tree T' iff T' has a full subtree T that satisfies the three conditions in Lemma 9. For simplicity, the automaton is defined over binary trees, where $D = \{0, 1\}$, but the definition can be easily extended to n-ary trees.

Let $\mathcal{A}_p = (\Gamma, Q, q_0, \delta, \beta)$. We define $Q = \{\mathsf{search}, \mathsf{cut}, \mathsf{wait}, \mathsf{reach}, \mathsf{visit}, \mathsf{err}\}$, $q_0 = \mathsf{search}$ and $\beta = \{\mathsf{visit}, \mathsf{wait}, \mathsf{cut}\}$. The states of the automaton can then be described as follows:

- search: In this state the automaton is searching for the root of the special subtree.
- cut: This represents a branch not taken.

- wait and reach: In these states the automaton has entered the subtree and is looking for nodes labeled with p.
- visit: In this state the automaton has just visited a node with label p in the subtree.
- err: This is an error state that is entered if there is a label higher than p in the subtree.

The transition function δ is defined as follows: For all $\rho = (X, j, M_i) \in \Gamma$,

1. For $q \in \{\mathsf{cut}, \mathsf{err}\}$, $\delta(q, \rho) = \{(q, q)\}$.
2. For $q = \mathsf{search}$

$$\delta(q,\rho) = \begin{cases} \{(\mathsf{search},\mathsf{cut}),(\mathsf{wait},\mathsf{cut})\} & \text{if } X = \{0\} \\ \{(\mathsf{cut},\mathsf{search}),(\mathsf{cut},\mathsf{wait})\} & \text{if } X = \{1\} \\ \{(\mathsf{search},\mathsf{cut}),(\mathsf{cut},\mathsf{search}),(\mathsf{wait},\mathsf{wait})\} & \text{if } X = \{0,1\} \end{cases}$$

3. For $q \in \{\mathsf{wait}, \mathsf{reach}, \mathsf{visit}\}$, if $j > p$ then $\delta(q, \rho) = \{(\mathsf{err}, \mathsf{err})\}$, if $j = p$ then

$$\delta(q,\rho) = \begin{cases} \{(\mathsf{visit},\mathsf{cut})\} & \text{if } X = \{0\} \\ \{(\mathsf{cut},\mathsf{visit})\} & \text{if } X = \{1\} \\ \{(\mathsf{visit},\mathsf{visit})\} & \text{if } X = \{0,1\} \end{cases}$$

and if $j < p$ then

$$\delta(q,\rho) = \begin{cases} \{(\mathsf{reach},\mathsf{cut})\} & \text{if } X = \{0\} \\ \{(\mathsf{cut},\mathsf{reach})\} & \text{if } X = \{1\} \\ \{(\mathsf{reach},\mathsf{wait}),(\mathsf{wait},\mathsf{reach})\} & \text{if } X = \{0,1\} \end{cases}$$

In the first stage, $\mathcal{A}_p$ guesses the location of the root of the special subtree T. While searching for this root, $\mathcal{A}_p$ remains in the state search. When it encounters the root, it enters the state wait for the first time. This starts the second stage, where $\mathcal{A}_p$ considers only marked nodes in T. In directions that correspond to a non-marked node, $\mathcal{A}_p$ moves to the state cut and remains there perpetually. From every marked node in T, $\mathcal{A}_p$ guesses a path to another marked node with label p, using the states wait and reach. It starts this search in state wait, moves to state reach immediately, remains there until it encounters a marked node with label p, and then moves to state visit. If there is no path from some node to another node with label p, all runs corresponding to the choice of T as subtree will eventually get stuck in reach. Thus, some run corresponding to T as the required subtree is accepting iff T satisfies the required conditions. ◀

▶ **Theorem 11.** *Let $\mathcal{L}$ be a library with width D, R be an exit control relation for $\mathcal{L}$, and α be an index function for $\mathcal{L}$. There exists a non-deterministic parity tree automaton (NPT) $\mathcal{B}$ such that, for all composers C over $\mathcal{L}$, $\mathcal{B}$ accepts $tree(C)$ iff C satisfies α and C is compatible with R. Consequently, $\mathcal{B}$ is non-empty iff $\mathcal{L}$ realizes α under R.*

Proof. We define $\mathcal{B} = \mathcal{B}_R \cap \mathcal{B}_\alpha$, where $\mathcal{B}_R$ is a safety tree automaton that accepts $tree(C)$ iff C is compatible with R, and $\mathcal{B}_\alpha$ is an NPT that accepts $tree(C)$ iff C satisfies α. Since the intersection of a safety automaton and an NPT is again an NPT, $\mathcal{B}$ is also an NPT.

Construction of $\mathcal{B}_R$: For simplicity, we define the automaton for the case $D = \{0, 1\}$, and note that the definition can be easily extended for arbitrary D. $\mathcal{B}_R = \{\mathcal{L}, \{\mathsf{start}\} \cup D, \mathsf{start}, \delta_R\}$, where δ_R is defined as follows: For all $M \in \mathcal{L}$,

- $\delta_R(\mathsf{start}, M) = \{(0, 1)\}$

- For $q \in D$, if $(q, M) \in R$ then $\delta_R(q, M) = \{(0, 1)\}$

Note that $\mathcal{B}_R$ has no transitions out of the states 0 and 1 iff the exit control relation R is violated. Thus $\mathcal{B}_R$ accepts $tree(C)$ iff C is compatible with R.

Construction of $\mathcal{B}_\alpha$: Let $\Gamma = LABELS(\mathcal{L})$ and let $\mathcal{A}_p = (\Gamma, Q, q_0, \delta, \beta)$ be the NBT defined in Lemma 10. We define $\mathcal{A}'_p = (\mathcal{L}, Q, q_0, \delta', \beta)$, where

$$\delta'(q, M_i) = \bigvee_{(X,j,M_i) \in LABELS(\mathcal{L})} \delta(q, (X, j, M_i))$$

While $\mathcal{A}_p$ accepts Γ-labeled D-trees, $\mathcal{A}'_p$ accepts $\mathcal{L}$-labeled D-trees. $\mathcal{A}'_p$ simply simulates $\mathcal{A}_p$ by using its larger transition function to guess the missing portion of the labels. We can characterize the regular trees accepted by $\mathcal{A}'_p$ as follows: for a composer C over $\mathcal{L}$, $\mathcal{A}'_p$ accepts $tree(C)$ iff there exists a choice function for C which has rank p.

Consider the automaton $\mathcal{A}'_\alpha$ whose language is the union of the language of each $\mathcal{A}'_p$, for all odd $p \leq \max(\alpha)$. Let C be a composer over $\mathcal{L}$. Then $\mathcal{A}'_\alpha$ accepts $tree(C)$ iff there exists a choice function for C that has an odd rank. Thus, by Theorem 7, $\mathcal{A}'_\alpha$ accepts $tree(C)$ iff C does not satisfy α. Finally, consider the automaton $\mathcal{B}_\alpha = \overline{\mathcal{A}'_\alpha}$, which is the complement of $\mathcal{A}'_\alpha$. Then $\mathcal{B}_\alpha$ accepts $tree(C)$ iff C satisfies α.

Since an NPT is nonempty iff it accepts a regular tree, and $\mathcal{L}$ realizes α under R iff some composer C over $\mathcal{L}$ satisfies α and C is compatible with R, therefore $\mathcal{B}$ is non-empty iff $\mathcal{L}$ realizes α under R. ◀

The NBT $\mathcal{A}'_p$ accepts $|D|$-ary trees and has $O(1)$ states, with an alphabet of size $|\mathcal{L}|$, so $\mathcal{A}'_\alpha$ is an NBT with $O(k)$ states, where $k = \max(\alpha)$. It follows that $\mathcal{B}_\alpha$ is a nondeterministic parity tree automaton (NPT) with $k^{O(k)}$ states and parity index $O(k)$ [12]. Also, $\mathcal{B}_R$ is a safety automaton with $O(|D|)$ states. Thus, their intersection $\mathcal{B}$ is an NPT with $|D|k^{O(k)}$ states and parity index $O(k)$, whose nonemptiness can be tested in time $|\mathcal{L}||D|^{O(k+|D|)}k^{O(k^2+k|D|)}$ [12]. We thus obtain the following:

▶ **Theorem 12.** *The embedded parity realizability problem is in EXPTIME.*

If an alternating tree automaton is nonempty, then it must accept some regular tree [12]. Given a regular tree accepted by $\mathcal{B}$, we can obtain a finite transducer that generates that tree. This transducer is a composer that realizes α under R. Thus, we also obtain a solution to the embedded parity synthesis problem.

▶ **Theorem 13.** *The embedded parity synthesis problem is in EXPTIME.*

The complexity of our solution is exponential in both k^2, where k is the highest parity index, as well as $|D|$, which is the number of exit states in each component. The exponential dependence on k is expected, as typical algorithms for solving parity games are exponential in the parity index, cf. [8]. Improving k^2 to k is an open challenge. It is also an open question whether the exponential dependence on $|D|$ can be avoided.

We remark that the embedded parity synthesis problem can be viewed as a 2-player partial information stochastic parity game. Informally, the game can be described as follows: The two players are the composer C and the environment E. The C player chooses components and the E player chooses paths through the components chosen by C. C cannot see the moves E makes inside a component. At the start C chooses a component M from the library $\mathcal{L}$. The turn passes to E, who chooses a sequence of inputs, inducing a path in M from its start state to some exit x in D. The turn then passes to C, which must choose some component M' in L and pass the turn to E and so on. As C cannot see the moves made by

E inside M, C cannot base its choice on the run of E in M, but only on the exit induced by the inputs selected by E and previous moves made by C. So C must choose the same next component M' for different runs that reach exit x of M. In general, different runs will visit different priorities inside M. This is a two-player stochastic parity game where one of the players does not have full information. If C has a winning strategy that requires a finite amount of memory, then we can use such a strategy to obtain a suitable finite composer that satisfies the index function α, thus solving the embedded parity synthesis problem. If C has no winning strategy or if every winning strategy requires infinite memory, then α is not realizable from the library L.

We also note that, when viewed in the framework of games, our result is a rare positive result for partial-information stochastic games. In general, 2-player partial information stochastic games are known to be undecidable even for co-Buchi objectives (and thus for parity objectives) [3].

5 Synthesis for DPW Specifications

Let A be a deterministic parity automaton (DPW), M be a probabilistic transducer and $\mathcal{L}$ be a library of components. We say A is a *monitor* for M (resp. $\mathcal{L}$) if the input alphabet of A is the same as the output alphabet of M (resp. $\mathcal{L}$). Let A be a monitor for M and let L_A be the language accepted by A. We say a strategy f for M is *winning* for the environment iff $\mu_f(L_A) < 1$, i.e., the output of M is rejected by A with positive probability. We say that M *satisfies* A if there exists no winning strategy for the environment.

▶ **Definition 14.** The *DPW probabilistic realizability problem* is: Given a library $\mathcal{L}$ and a DPW A that is a monitor for $\mathcal{L}$, decide whether there exists a composer C over $\mathcal{L}$, such that $\mathcal{T}_C$ satisfies A. If such a composer exists, we say that $\mathcal{L}$ *realizes* A. The *DPW probabilistic synthesis problem* is to find such a composer C if it exists.

We transform this problem into a version of the embedded parity problem solved in the previous section. Let $A = (\Sigma_O, Q_A, s_0, \delta_A, \alpha_A)$ be a DPW and $M = (\Sigma_I, \Sigma_O, Q_M, q_0, \delta_M, F, L)$ be a probabilistic transducer. For $s \in Q_A$, we denote by $M \times A_s$, the probabilistic transducer $(\Sigma_I, \Sigma_O, Q_M \times Q_A, (q_0, s), \delta, F \times Q_A, L')$, where $\delta((q, s'), a)(q', s'') = \delta_M(q, a)(q')$ if $s'' = \delta_A(s', L(q))$ and 0 otherwise. Given a library $\mathcal{L}$ with width D, we define the *augmented library* $\mathcal{L}_A = \{M \times A_s : M \in \mathcal{L}, s \in Q_A\}$. The width of $\mathcal{L}_A$ is $D \times Q_A$. We define the exit control relation $R_A \subseteq D \times Q_A \times \mathcal{L}_A$ for $\mathcal{L}_A$ as follows: for all $i \in D$, $s \in Q_A$, $M \in \mathcal{L}$, we have $(i, s, M \times A_s) \in R_A$. We also extend α_A to $\mathcal{L}_A$ as follows: for $(q, s') \in Q_M \times Q_A$, $\alpha_A(q, s') = \alpha_A(s')$. Thus α_A is an index function for $\mathcal{L}_A$.

Our first step is to treat this augmented library as a new library and solve the embedded parity synthesis problem for $\mathcal{L}_A$ with α_A as the index function and R_A as the exit control relation. This gives us a tree automaton that accepts $\mathcal{L}_A$-labeled $(D \times Q_A)$-trees and that is empty iff $\mathcal{L}_A$ does not realize α_A under R_A. Later, we show how to transform this automaton into another that accepts $\mathcal{L}$-labeled D-trees and is empty iff $\mathcal{L}$ does not realize A. Since, by definition, $\mathcal{L}_A$ bijectively maps to $\mathcal{L} \times Q_A$, we find it convenient to use labels from $\mathcal{L} \times Q_A$ in place of $\mathcal{L}_A$. We now define a composer for the augmented library. The states of the composer are pairs of the form (M, s), where s is a monitor state and M represents an instance of a component from $\mathcal{L}$. A *composer* for $\mathcal{L}_A$, is a deterministic transducer $C = (D \times Q_A, \mathcal{L} \times Q_A, \mathcal{M} \times Q_A, (\mathsf{M}, s), \Delta, \lambda)$. The following lemma follows directly from

Theorem 11[1].

▶ **Lemma 15.** *Let $\mathcal{L}$ be a library and A be a DPW that is a monitor for $\mathcal{L}$. There exists an NPT $\mathcal{B}$ that accepts a regular tree T iff $T = tree(C)$ for some composer C over $\mathcal{L}_A$ such that $\mathcal{T}_C$ satisfies α_A and C is compatible with R_A.*

Given a composer C over a library $\mathcal{L}$ and a monitor A for $\mathcal{L}$, we can extend C to a composer over the augmented library $\mathcal{L}_A$.

▶ **Definition 16** (Augmented Composer)**.** Let $\mathcal{L}$ be a library and A be a monitor for $\mathcal{L}$. Let $C = (D, \mathcal{L}, \mathcal{M}, \mathsf{M}_0, \Delta, \lambda)$ be a composer over $\mathcal{L}$. The *augmentation* of C by A, denoted C_A, is a composer over $\mathcal{L}_A$ such that $C_A = (D \times Q_A, \mathcal{L} \times Q_A, \mathcal{M} \times Q_A, (\mathsf{M}_0, s_0), \Delta', \lambda')$, where

1. For all $s \in Q_A$, $\mathsf{M} \in \mathcal{M}$, $\lambda'(\mathsf{M}, s) = (\lambda(\mathsf{M}), s)$.
2. For all $i \in D$, $\mathsf{M} \in \mathcal{M}$ and $s, s' \in Q_A$, $\Delta((\mathsf{M}, s), (i, s')) = (\Delta(\mathsf{M}, i), s')$.

We say C_A is an augmented composer. While a composer only keeps track of the transfer of control between components, the augmented composer also keeps track of the state of the monitor before and after the control is transferred. To go from augmented composers to composers, we use techniques from synthesis with incomplete information [10]. We start by describing a relation between $tree(C)$ and $tree(C_A)$. First we need to introduce some convenient notation.

Let X, Y and Z be finite sets. For a Z-labeled $(X \times Y)$-tree $\langle T, V\rangle$, we denote by $xray(Y, \langle T, V\rangle)$, the $(Z \times Y)$-labeled $(X \times Y)$-tree $\langle T, V'\rangle$ in which each node is labeled by both its direction in Y and its labeling in $\langle T, V\rangle$. We define operators $hide_Y$ and $wide_Y$. The operator $hide_Y : (X \times Y)^* \to X^*$ replaces each letter $x \cdot y$, where $x \in X$ and $y \in Y$, by the letter x. The operator $wide_Y$ maps Z-labeled X-trees to Z-labeled $(X \times Y)$-trees as follows: $wide_Y(\langle X^*, V\rangle) = \langle (X \times Y)^*, V'\rangle$, where for each node $w \in (X \times Y)^*$, we have $V'(w) = V(hide_Y(w))$.

▶ **Lemma 17.** *Let $\mathcal{L}$ be a library and A be a monitor for $\mathcal{L}$. Let C be a composer over $\mathcal{L}$ and C_A be the augmentation of C by A. Then $tree(C_A) = xray(Q_A, wide_{Q_A}(tree(C)))$.*

▶ **Theorem 18.** *Let $\mathcal{L}$ be a library and A be a monitor for $\mathcal{L}$. Let C be a composer over $\mathcal{L}$ and C_A be the augmentation of C by A. Then C satisfies A iff C_A satisfies α_A.*

Given a library $\mathcal{L}$ and monitor A, we can solve the embedded realizability problem for the augmented library $\mathcal{L}_A$ to obtain a regular tree T, where $T = tree(C)$ for some composer C over $\mathcal{L}_A$ such that C satisfies α_A. Then the tree $T' = xray(Q_A, wide_{Q_A}(tree(C)))$ is also regular, so $T' = tree(C')$ for some composer C' over $\mathcal{L}$. Now we would like to use C' to solve the DPW realizability problem, but C' is only guaranteed to satisfy A if C is the augmentation of C' by A. Therefore, to solve the DPW realizability problem, we have to obtain an automaton that accepts a tree $T' = tree(C')$ if the augmentation of C' by A satisfies α_A.

▶ **Theorem 19.** *Let X, Y and Z be finite sets. Given an alternating automaton $\mathcal{B}$ over $(Z \times Y)$-labeled $(X \times Y)$-trees, we can construct an alternating automaton $\mathcal{B}'$ over Z-labeled X-trees such that $\mathcal{B}'$ accepts a labeled tree $\langle X^*, V\rangle$ iff $\mathcal{B}$ accepts $xray(Y, wide_Y(\langle X^*, V\rangle))$. Further, $\mathcal{B}$ and $\mathcal{B}'$ have the same acceptance condition and $|\mathcal{B}'| = O(|\mathcal{B}|)$.*

[1] Note that even with the slightly modified definition of composer, the results of the previous section still apply because a pair $(M, s) \in \mathcal{L} \times Q_A$ still uniquely identifies an element of $\mathcal{L}_A$.

Given an alternating automaton $\mathcal{B}$, let $narrow_Y(\mathcal{B})$ denote the corresponding automaton constructed in Theorem 19.

▶ **Theorem 20.** *Let $\mathcal{L}$ be a library and A be a monitor for $\mathcal{L}$. Then there exists an alternating parity tree automaton (APT) $\mathcal{B}$ such that, for all composers C over $\mathcal{L}$, $\mathcal{B}$ accepts $tree(C)$ iff C satisfies A. Consequently, $\mathcal{B}$ is non-empty iff $\mathcal{L}$ realizes A.*

Proof. Let $A = (\Sigma_O, Q_A, s_0, \delta_A, \alpha_A)$. Let $\mathcal{B}'$ be the NPT that accepts $tree(C')$ iff C' satisfies α_A and C' is compatible with R_A, for all composers C' over $\mathcal{L}_A$. Such a $\mathcal{B}'$ exists by Lemma 15. Let $\mathcal{B} = narrow_{Q_A}(\mathcal{B}')$. We show that $\mathcal{B}$, which is an APT, is the required automaton.

Let C be a composer over $\mathcal{L}$. By Theorem 18, C satisfies A iff C_A satisfies α_A. Therefore, $\mathcal{B}'$ accepts $tree(C_A)$ iff C satisfies A. By Lemma 17, $tree(C_A) = xray(Q_A, wide_{Q_A}(tree(C)))$, and by Theorem 19, $\mathcal{B}$ accepts a tree T iff $\mathcal{B}'$ accepts $xray(Q_A, wide_{Q_A}(T))$. Thus, $\mathcal{B}$ accepts $tree(C)$ iff C satisfies A. Since an APT is nonempty iff it accepts a regular tree, and $\mathcal{L}$ realizes A iff some composer C over $\mathcal{L}$ satisfies A, hence $\mathcal{B}$ is non-empty iff $\mathcal{L}$ realizes A. ◀

Each transducer in the augmented library $\mathcal{L}_A$ has a set of final states of size $|D||Q_A|$. Thus the automaton $\mathcal{B}'$ has size exponential in both $|D|$ and $|Q_A|$. The translation from $\mathcal{B}'$ to $\mathcal{B}$ adds no blowup, but $\mathcal{B}$ is an APT, while $\mathcal{B}'$ is an NPT. Since emptiness for an alternating parity tree automaton can be checked in time exponential in the size of the automaton [12], therefore $\mathcal{B}$ can be be checked for emptiness in time doubly exponential in $|D|$ and $|Q_A|$.

▶ **Theorem 21.** *The DPW probabilistic realizability problem is in 2EXPTIME.*

Again, if an alternating tree automaton is nonempty, then it must accept some regular tree [12], and given a regular tree accepted by $\mathcal{B}$, we can obtain a finite transducer that generates that tree. This transducer is a composer that realizes A. Thus, we also obtain a solution to the DPW probabilistic synthesis problem.

▶ **Theorem 22.** *The DPW probabilistic synthesis problem is in 2EXPTIME.*

The doubly exponential upper bound for our solution can be viewed as follows: we inherit one exponential from the embedded parity solution and the second exponential is introduced by the use of an APT to deal with incomplete information. It is an open question whether the second exponential can be avoided.

6 Discussion and Future Work

Component-based synthesis seeks to build systems that satisfy a given specification using pre-existing components. This contrasts with classical synthesis, where the aim is to build a system from scratch. The component-based approach is closer in spirit to how systems are built in the real world. In this paper, we generalize the component-based synthesis problem to a probabilistic setting. Our components are modeled as probabilistic transducers and the specification is given as a deterministic parity automaton. The composition itself is described by a deterministic transducer, called a *composer*, which governs the transitions between components.

We break the problem down in two stages. First we solve a simpler version, which we call the *embedded parity synthesis problem*, where the specification is embedded as parities in the components themselves. Our solution combines techniques from Markov chain analysis and automata theoretic verification. Then we show how to solve the more general case of a separate specification, which we call the *DPW probabilistic synthesis problem*, by reducing it to the simpler case using techniques from synthesis with incomplete information.

We show that the embedded parity synthesis problem is in EXPTIME and the DPW probabilistic synthesis problem is in 2EXPTIME. The question of tighter lower and upper bounds we leave for future work. In particular, it is an open question whether the DPW probabilistic synthesis problem is in EXPTIME. Another line of work is suggested by the possibility of probabilistic composers. While we do not know how to synthesize probabilistic composers, we do know that a direct reduction to the deterministic case will not work as probabilistic composers are more expressive.

References

1 C. Baier, M. Größer, M. Leucker, B. Bollig, and F. Ciesinski. Controller synthesis for probabilistic systems. In *Proc. IFIP TCS'04*, pages 493–506. Kluwer, 2004.

2 D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Proc. ICSOC'03*, LNCS 2910, pages 43–58. Springer, 2003.

3 K. Chatterjee and L. Doyen. The complexity of partial-observation parity games. In *Proc. LPAR'10*, LNCS 6397. Springer, 2010.

4 K. Chatterjee, M. Jurdzinski, and T. A. Henzinger. Simple stochastic parity games. In *Proc. CSL'03*, LNCS 2803, pages 100–113. Springer, 2003.

5 C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events. In *Proc. ICALP'90*, LNCS 443, pages 336–349. Springer, 1990.

6 C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42:857–907, 1995.

7 L. de Alfaro and T.A. Henzinger. Interface-based design. In *Engineering Theories of Software-intensive Systems*, NATO Science Series: Mathematics, Physics, and Chemistry 195, pages 83–104. Springer, 2005.

8 M. Jurdzinski. Small progress measures for solving parity games. In *Proc. STACS'00*, LNCS 1770, pages 290–301. Springer, 2000.

9 J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Van Nostrad, 1960.

10 O. Kupferman and M.Y. Vardi. Synthesis with incomplete informatio. In *2nd Int. Conf. on Temporal Logic*, pages 91–106. Kluwer, 1997.

11 Y. Lustig and Moshe Y. Vardi. Synthesis from component libraries. In *Proc. FOSSACS'09*, LNCS 5504, pages 395 – 409. Springer, 2009.

12 D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995.

13 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.

14 S. Schewe. Synthesis for probabilistic environments. In *Proc. ATVA'06*, LNCS 4218. Springer, 2006.

15 J. Sifakis. A framework for component-based construction extended abstract. In *Proc. 3rd Int. Conf. on Software Engineering and Formal Methods*, pages 293–300. IEEE, 2005.

16 M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. FOCS'85*, pages 327–338. IEEE, 1985.

17 M.Y. Vardi. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In *Formal Methods for Real-Time and Probabilistic Systems*, LNCS 1601, pages 265–276. Springer, 1999.

Synthesizing Reactive Programs*

P. Madhusudan

University of Illinois at Urbana-Champaign, USA
madhu@cs.illinois.edu

Abstract

Current theoretical solutions to the classical Church's synthesis problem are focused on synthesizing *transition systems* and not *programs*. Programs are compact and often the true aim in many synthesis problems, while the transition systems that correspond to them are often large and not very useful as synthesized artefacts. Consequently, current practical techniques first synthesize a transition system, and then extract a more compact representation from it. We reformulate the synthesis of reactive systems directly in terms of program synthesis, and develop a theory to show that the problem of synthesizing programs over a fixed set of Boolean variables in a simple imperative programming language is decidable for regular ω-specifications. We also present results for synthesizing programs with recursion against both regular specifications as well as visibly-pushdown language specifications. Finally, we show applications to program repair, and conclude with open problems in synthesizing distributed programs.

1998 ACM Subject Classification I.2.2 Automatic Programming; F.3.1 Specifying and Verifying and Reasoning about Programs; F.4.3 Formal Languages

Keywords and phrases Program synthesis, boolean programs, automata theory, temporal logics

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.428

1 Introduction

The synthesis problem for reactive systems is a classical problem in computer science, and stems from a problem posed by Church in 1957 on synthesizing digital circuits from specifications written in a restricted logic of arithmetic [6]. This problem was solved first by Büchi and Landweber in 1969 [5] (see [25] for an account of the history of this problem). The 70s saw the emergence of the elegant theory of *automata on infinite trees* by Rabin [19], which has now been well-studied and honed into a beautiful theory that underlies many of the decidability results in logic and automata theory [24, 8]. Coupled with the temporal-logic to automata connection on words [16, 27, 20], tree-automata theory gives the most elegant solution to Church's problem: compile the specification into a *deterministic* parity automaton on infinite words [20], using this build a parity tree-automaton that accepts the trees that correspond to *strategies* for the system to generate outputs for inputs so that all paths in the tree are accepted by the specification automaton, and, finally, check the emptiness of the tree automaton and build a finite-state transition system from a regular tree that's accepted by this automaton [17].

The synthesis problem has received a lot of attention in recent years, both in theory as well as in practice. Theoretical approaches include extensions to branching time specifications [11, 13], the very non-trivial problem of synthesizing *distributed systems* [18, 14, 13], and synthesis with incomplete information where the environment and system may not have perfect information about the state of each other [10].

* This research was partially supported by NSF CAREER award #0747041.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 428–442

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

Most of the current theoretical techniques in synthesis are geared towards designing *transition systems*. In other words, the algorithms for synthesis in the end output transition systems that meet the specification. While transition systems are appropriate for defining semantics of systems, systems are seldom designed by explicitly describing their transition systems. Systems are instead designed using high-level and succinct representations such as *programs*.

The extensive literature on synthesizing transition systems does not help in building directly the compact programs we seek. Current practical techniques build a transition system (or an automaton that depicts several different strategies for winning), and implement it in a symbolic manner such as a program. Several such approaches currently exist; for instance, the approach in [15] builds automata from which a symbolic algorithm using memory variables is extracted and the work by Bloem et al [4] reports on practical ways of synthesizing PSL circuit code from symbolic BDD representations of the solution.

Consider, for instance, the problem of synthesizing a data-structure to maintain subsets of a fixed set of n elements, where the environment is allowed to maintain the set by adding or removing elements from it, and querying the set for membership. It is easy to see that a program for implementing it can be written that uses n Boolean variables, one for each element, that tracks whether that element belongs to the set or not. Furthermore, a short while-program (of length about $O(n\ log\ n)$) can be written. On the other hand, a transition system realizing the specification will necessarily be exponential in n. For $n = 50$, a transition system with 2^{50} states is not very useful as a synthesized artefact, while a program that's about 150 lines of code is. One can, of course, construct a particular transition system and then try to synthesize a program from it (as in the works cited above), but these are not guaranteed to yield small programs.

Another drawback (and a subtle one) in current synthesis techniques is that the systems that are synthesized depend on *how* specifications are written, rather than only its semantics. For instance, assume φ_1 and φ_2 are two specifications that are syntactically different but semantically equivalent. Then the current synthesis algorithms based on tree automata will produce *different* sets of (finite) transition systems for these specifications. The reason is that for a given specification, current synthesis algorithms build an automaton that accepts unfoldings of transitions systems that satisfy the specification. Though these automata for φ_1 and φ_2 accept precisely the same set of trees, the automata themselves are different, as they depend on the *syntax* of the specifications (i.e., on φ_1 and φ_2). Consequently, the non-emptiness algorithm for these automata can synthesize different transition systems for the two semantically-equivalent specifications. Intuitively, the synthesized finite-state transition system not only encodes a correct algorithm, but also an elaborate *proof* as to why it meets the specification, and this proof grows with the way the specification is formulated. In particular, the more complex the same specification is written, the more complex will be the synthesized transition system!

There is some work in the literature (see [21]) that addresses the above problem and can synthesize transition systems (but not programs) that are within a bound, independent of the specification. However, we do not know of any work on specification-syntax agnostic synthesis that works in the unbounded setting.

The main contribution of this paper is a theory of synthesis where imperative programs (written in a particular syntax, and over a fixed set of Boolean variables) are first-class objects and where the synthesis algorithms directly synthesize programs that meet reactive regular ω-specifications. We lay out this theory of synthesizing imperative programs, showing two main results: (a) that imperative program synthesis is decidable against regu-

lar specifications, and (b) that synthesis of imperative programs with recursive functions is decidable against both regular specifications as well as visibly-pushdown specifications.

The technical results are automata-theoretic, are geared towards synthesizing programs directly, and use two-way alternating ω-automata working on *finite* trees that represent programs. In particular, the synthesis algorithms in this paper build tree automata on finite trees that accept *all* programs satisfying a specification, and hence do not depend on the way the specification is written. We can therefore pull out a program that accords to our needs (for example, we can pull out the smallest program satisfying the specification, for some notion of length of a program).

There is a flurry of research in the programming languages community in the last few years on *program synthesis* [9, 23, 22]. In general, these algorithms are aimed at practical synthesis approaches towards solving standard algorithmic problems (such as sorting, Strassen's multiplication algorithm, Excel scripts, etc.). In these papers, the prevailing theme is to *fix* a template for the program, and use SAT and SMT solvers to find a program matching the template and simultaneously a proof (which also comes with a template) that proves the program correct. In these algorithms, the search space of programs for synthesis is *finite*, and the focus is on efficiency, programmability, and usability. Hence a theory of synthesizing programs, albeit finite-state programs, seems worthy of study. In this context, this paper provides a sound and complete procedure for synthesizing Boolean programs of arbitrary length that satisfy a specification.

The paper is structured as follows: Section 2 defines imperative programs without recursion and regular specifications, while Section 3 introduces the automata theory on trees that we will use. Section 4 lays down the results for synthesizing non-recursive programs, and Section 5 shows how to extend this to synthesize recursive programs against regular as well as visibly pushdown specifications. Section 6 concludes with a discussion of applications of our results to program repair, and open problems in distributed program synthesis.

2 Programs and regular specifications

We define in this section the class of imperative programs that we work with, and also define the class of regular specifications against which we synthesize programs.

Let us fix two numbers $N_I, N_O \in \mathbb{N}$. We will design programs that in every round take N_I bits as input and output N_O bits.

Programs are parameterized over a finite set of Boolean variables B that it uses (naturally, we assume $|B| \geq max\{N_I, N_O\}$). The class of programs over B is given as follows (where b, b_i range over variables in B, and $\vec{b}$ stands for a vector of variables in B):

$$
\begin{array}{lcl}
\langle stmt \rangle & ::= & \langle stmt \rangle ; \langle stmt \rangle \mid \textbf{skip} \mid b := \langle expr \rangle \mid \textbf{input } \vec{b} \mid \textbf{output } \vec{b} \\
 & & \textbf{if } (\langle expr \rangle) \textbf{ then } \langle stmt \rangle \textbf{ else } \langle stmt \rangle \mid \textbf{while } (\langle expr \rangle) \{\langle stmt \rangle\} \\
\langle expr \rangle & ::= & b \mid \textbf{tt} \mid \textbf{ff} \mid \langle expr \rangle \vee \langle expr \rangle \mid \neg \langle expr \rangle
\end{array}
$$

We assume that all input and output statements have tuples of width N_I and N_O, respectively.

The semantics of a reactive program is the natural one. The "**input** $\vec{b}$" statement takes (reactively) an input in $\{0,1\}^{N_I}$ from the environment and stores it in the variables $\vec{b}$, while "**output** $\vec{b}$" outputs the values of the variables in $\vec{b}$. The program can execute any number of internal steps between an input and an output statement (though synthesized programs will have the property that the program eventually does produce an output). During this internal computation, the program can manipulate its variables using assignments, conditionals, and iteration. Note that programs are, of course, finite in length, though they can

(and typically will) interact with their environment reactively and infinitely often.

Representing programs using finite trees: We will represent programs as finite trees. Intuitively, the *bracketing* of blocks of code (as defined by the statements under conditionals and while-loops) can be nested arbitrarily, and hence we need trees to capture roughly the parse-trees of programs according to the grammar given above.

For brevity, we will use binary trees (with every node having zero, one, or two children) and represent them using terms. A term $f(t_1, t_2)$ corresponds to a tree with f as the label of the root, and with the trees corresponding to t_1 and t_2 as the left and right subtrees of the root; a unary term $g(t)$ corresponds to a tree with g as the label of the root, and where the root has only one child (say the left child), and the subtree at this child is isomorphic to the tree associated with t.

The tree associated with a program P is $(root, tree(p))$, where $tree$ is inductively defined as follows:

$$
\begin{array}{rcl}
tree(b) & = & b \\
tree(\mathbf{tt}) & = & \mathbf{tt} \\
tree(\mathbf{ff}) & = & \mathbf{ff} \\
tree(\varphi_1 \vee \varphi_2) & = & \vee\ (tree(\varphi_1),\ tree(\varphi_2)) \\
tree(\neg\varphi) & = & \neg\ (tree(\varphi_1))
\end{array}
\qquad
\begin{array}{rcl}
tree(s; s') & = & ;\ (tree(s),\ tree(s')) \\
tree(\mathbf{skip}) & = & \mathbf{skip} \\
tree(\mathbf{input}\ \vec{b}) & = & \mathbf{input}\ \vec{b} \\
tree(\mathbf{output}\ \vec{b}) & = & \mathbf{output}\ \vec{b} \\
tree(b := e) & = & assign\text{-}b\ (tree(e)) \\
tree(\mathbf{if}\ (e)\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2) & = & \mathbf{if}\ (tree(e), \\
 & & \mathbf{then}(tree(s_1),\ tree(s_2))) \\
tree(\mathbf{while}\ (e)\{s\}) & = & \mathbf{while}\ (tree(e),\ tree(s))
\end{array}
$$

A program over a set of Boolean variables B is hence encoded as a binary tree over the alphabet:

$$
\begin{array}{c}
\Sigma = \{root, \neg, \vee, ;, \mathbf{if}, \mathbf{then}, \mathbf{while}\} \cup B \cup \{assign\text{-}b \mid b \in B\} \\
\cup \{\mathbf{input}\ \vec{b} \mid \vec{b} \in B^{N_I}\} \cup \{\mathbf{output}\ \vec{b} \mid \vec{b} \in B^{N_O}\}
\end{array}
$$

The tree automata we build will accept such trees that represent programs that satisfy the given specification. Of course, the set of trees corresponding to *all* programs over a particular *finite* set of Boolean variables B is regular; we skip this proof, as it follows pretty much from the fact that the set of parse trees of any context-free grammar is regular.

▶ **Lemma 1.** *Let B be a finite set of variables. Then the set of trees corresponding to programs over variables B is regular.* ◀

Regular specifications:

A (linear-time) specification over (N_I, N_O) is a subset of ω-sequences $L \subseteq (\{0,1\}^{N_I+N_O})^\omega$ that depicts the correct infinite sequences of input-output behavior allowed by the specification. A specification L is said to be *regular* if there is a *non-deterministic Büchi automaton* over the alphabet $\{0,1\}^{N_I+N_O}$ that precisely accepts L (we don't define word automata in this paper; we refer the reader to [24]).

We will assume that regular specifications L are given by a non-deterministic automaton that accepts the set of sequences *not* in L, i.e., by a non-deterministic Büchi automaton accepting $\overline{L} = (\{0,1\}^{N_I+N_O})^\omega \setminus L$. This is without loss of generality as regular languages are effectively closed under complement.

For any linear-time temporal logic (LTL) specification φ over a set of $N_I + N_O$ propositions can be seen as defining a regular specification $L_\varphi = \{\alpha \in (\{0,1\}^{N_I+N_O})^\omega \mid \alpha \models \varphi\}$.

Furthermore, given an LTL specification φ, we can construct a Büchi automaton A accepting $\overline{L}$ in time exponential in $|\varphi|$ and whose size is exponential in $|\varphi|$, by building the automaton accepting the models of $\neg\varphi$, using the now-classic temporal-logic–automata connection [27].

We will work with regular ω-specifications given by automata accepting $\overline{L}$ in the sequel, derive the complexity results for synthesis in terms of the size of this automaton, and, as a corollary, derive the complexity of synthesis for LTL specifications.

3 Trees and Alternating automata

We use labeled binary *finite* trees throughout this paper. Given a finite set of labels Σ, an Σ-labeled tree is a pair $T = (V, \lambda)$, where $V \subseteq \{L, R\}^*$ that is finite and prefix closed, and $\lambda : V \to \Sigma$. The edges of the tree are implicit: for every $v \in V$, if $v.L \in V$, then $v.L$ is the left-child of v, and if $v.R \in V$, then $v.R$ is the right-child of v; the node ϵ is the root of the tree. The function λ assigns a label in Σ to each node of the tree.

For convenience, let us overload the concatenation operator so that for any $v \in \{L, R\}^*$, $v.U = v'$ if $v'.L = v$ or $v'.R = v$; i.e., $v.U$ refers to the parent of v in the tree, obtained by going *up* from v.

Non-deterministic finite automata on trees are the classic top-down tree-automata, with different transition functions defined for nodes that have a left-child only, a right-child only, or has both children. We refer the reader to a textbook on tree automata for details [7]; we fix here only a brief definition and notation.

A *non-deterministic finite tree automaton* on Σ-labeled trees is a structure $\mathcal{A} = (Q, q_0, \delta_L, \delta_R, \delta_{LR}, F)$, where Q is a finite set of states, $q_0 \in Q$, $\delta_L, \delta_R : Q \times \Sigma \to 2^Q$, $\delta_{LR} : Q \times \Sigma \to 2^{Q \times Q}$, and $F \subseteq Q$.

A run of such a tree automaton on a finite tree (V, λ) is a Q-labeled tree (V, ρ) where: $\rho(\epsilon) = q_0$, and for every $v \in V$, (i) if v has a left-child but no right-child, then $\rho(v.L) \in \delta_L(q, \lambda(v))$; (ii) if v has a right-child but no left-child, then $\rho(v.R) \in \delta_R(q, \lambda(v))$; and (iii) if v has both children, then $(\rho(v.L), \rho(v.R)) \in \delta_{LR}(q, \lambda(v))$

A run (V, ρ) is accepting if for every leaf v of V, $\rho(v) \in F$. A tree is accepted by the automaton if there is an accepting run on it. The language of the tree automaton is the set of all Σ-labeled trees accepted by it.

Two-way alternating ω-automata on finite trees:
We now define two-way alternating ω-automata on finite trees. This is a bit unusual; ω-automata are usually defined on *infinite trees*, not on finite ones. However, we will deal with only accepting finite-trees in this paper, which will be used to encode programs (which have a finite description, of course). However, in order to simulate these programs on ω-length sequences of inputs, we would need tree automata working on finite trees for infinitely many steps, going up and down the tree.

For any set S, let $\mathcal{B}^+(S)$ denote the set of all positive Boolean formulas over S; i.e., the set defined by the grammar:

$$\varphi ::= \mathit{true} \mid \mathit{false} \mid s \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$

where $s \in S$.

A *two-way alternating Büchi tree automaton* over Σ-labeled trees, is a tuple $A = (Q, q_0, \delta_L, \delta_R, \delta_{LR}, \delta_\emptyset, F)$, where Q is a finite set of states, $q_0 \in Q$, $F \subseteq Q$, and where:

- $\delta_L : Q \times \Sigma \times \{L, D\} \to \mathcal{B}^+(Q \times \{L, U\})$
- $\delta_R : Q \times \Sigma \times \{R, D\} \to \mathcal{B}^+(Q \times \{R, U\})$

- $\delta_{LR} : Q \times \Sigma \times \{L, R, D\} \to \mathcal{B}^+(Q \times \{L, R, U\})$
- $\delta_{\emptyset} : Q \times \Sigma \times \{D\} \to \mathcal{B}^+(Q \times \{U\})$

Intuitively, $\delta_{LR}(q, a, dir) = \varphi$ denotes the actions the automaton can take when in state q, reading a node n whose label is a, and when the *last* move it did is given by dir, where dir=L (dir=R) means that in the last move the tree automaton came *up* from the left child (respectively, right child) of n, and dir=D means that in the last move the tree automaton came *down* from the parent of n. The tree automaton, at such a point, is allowed to choose any Boolean valuation of the set $(Q \times \{L, R, U\})$ that satisfies the formula φ, and for every (q', g) that it sets to true, it must pass a copy of itself in state q' along the direction g, where $g = L, R, U$ is interpreted as left-child, right-child, and up to the parent, respectively. The transitions δ_L and δ_R (for nodes with only a left-child or only a right-child) and $\delta_{\emptyset}$ (for leaves of the tree) are similarly interpreted. By convention, we assume that at the beginning, the tree automaton starts at the root with the last move set to dir=D. The tree automaton accepts the tree if all its branches formed by the implicit infinite tree it defines by propagating states meet the set F infinitely often.

Two-way alternating co-Büchi automata are similarly defined; here the tree automaton accepts if all its branches meet F only *finitely* often.

Semantics of two-way alternating automata:
Formally, let us define the acceptance of a tree by an automaton $A = (Q, q_0, \delta_L, \delta_R, \delta_{LR}, \delta_{\emptyset}, F)$ using a game, played between two players, the automaton-player (player 0) and the path-finder player (player 1) (we can also equivalently define acceptance using infinite trees).

A *finite-state two-player Büchi arena* is a tuple $G = (P_0, P_1, E, p_0, F)$ where P_0 and P_1 are two finite disjoint sets representing the positions from where players 0 and 1 play, respectively, $E \subseteq (P_0 \times P_1) \cup (P_1 \times P_0)$ is a set of edges that defines a bipartite graph over P_0 and P_1, $p_0 \in P_0$ is the initial position, and $F \subseteq P_0 \cup P_1$ is a set of Büchi positions.

A *strategy for player* 0 is a function $f_0 : (P_0P_1)^* \to P_1$, such that for any $\sigma \in (P_0P_1)^*$ and $p \in P_1$, $(p, f_0(\sigma.p)) \in E$. In other words, f_0 encodes a strategy for player 0 to choose a successor vertex after any finite sequence of moves that is a partial play in the game.

A *play* is a finite or infinite path in the graph defined by the arena, and denoted by a sequence in $\{p_0\}.(P_1P_0)^*(\epsilon + P_1) \cup \{p_0\}.(P_1P_0)^{\omega}$. A maximal play is a play that cannot be extended (an ω-length play is maximal; a finite-length play is maximal only if the final vertex has no out-going edges). A play σ *conforms* to a strategy f_0 for player 0 if for every proper prefix $\sigma' \in (P_0P_1)^*P_0$ of σ, $\sigma' f(\sigma')$ is also a prefix of σ. A strategy for player 0, f_0, is said to be winning if for all maximal plays σ that conform to f_0, σ is not finite and some position in F occurs infinitely often in σ. We say that player 0 wins the game on the arena if it has a winning strategy.

Intuitively, a strategy for player 0 is winning if along any play conforming to the strategy player 1 gets "stuck" (cannot make a move) or the play is infinite and meets the Büchi final state set infinitely often.

We can now define when a two-way alternating automaton $A = (Q, q_0, \delta_L, \delta_R, \delta_{LR}, \delta_{\emptyset}, F)$ accepts a tree (V, λ). Let us define the arena corresponding to A and the tree as (P_0, P_1, E, p_0, F') where:

- $P_0 = (V \times Q \times \{L, R, D\})$
- $P_1 = (V \times 2^{Q \times \{L,R,U\}})$

- E contains the following edges:
  - An edge from (v, q, dir) to (v, S) iff setting S to true and the other elements in $Q \times \{L, R, U\}$ to false satisfies $\delta_h(q, \lambda(v), dir)$, where $h{=}L$ if v has only a left-child, $h{=}R$ if v has only a right-child, and $h{=}LR$ if v has two children, and $h = \emptyset$ if v is a leaf.
  - An edge from (v, S) to (v', q', dir) iff $(q', g) \in S$ and $v' = v.g$ and either $g \in \{L, R\}$ and $dir = D$, or $g = U$ and $v'.dir = v$

The initial position is $p_0 = \{(\epsilon, q_0, D)\}$. The set of Büchi states is $F' = V \times F \times \{L, R, D\}$.

Then the automaton $\mathcal{A}$ *accepts the tree* (V, λ) if player 0 has a winning strategy on the corresponding arena. The language of the automaton $\mathcal{A}$ is the set of trees accepted by it.

We can similarly define *co-Büchi* automata; here player 0 wins iff if the set F is met only *finitely* often.

The size of a two-way alternating automaton is the length of its description encoded as a string.

Two-way alternating automata to one-way non-deterministic automata: It is well-known that two-way alternating tree automata can be converted to non-deterministic tree automata with an exponential blow-up in the state-space (see [26] for example, where such a construction is shown for automata on infinite trees). We can do a similar construction to convert alternating Büchi or co-Büchi tree automata over finite trees to an equivalent non-deterministic automaton over finite trees, with an exponential blow-up. We omit the proof here; a gist of the proof can be found in [12]. Consequently two-way Büchi and co-Büchi alternating tree automata on finite trees capture only regular tree languages, and their emptiness problem can be decided in exponential time.

As an auxiliary notation, in the sequel, we sometimes write transitions of the form $\delta(q, a, dir) = (q', LR)$, where we mean by (q', LR) that the automaton passes the state q' to the right-child of the left-child of the current node. We do this for brevity; such transitions can easily be converted to standard transitions using intermediate states.

4 Synthesizing Reactive Programs

In this section, we prove our first main result: for any regular specification and a set of Boolean variables B, we can build a tree automaton that precisely accepts the class of all tree-encodings of programs over B that satisfy the specification. By checking emptiness of this tree automaton, we can synthesize programs that satisfy the specification, and in particular, synthesize the smallest programs satisfying the specification.

Let us fix input and output arities $N_I, N_O \in \mathbb{N}$ for the rest of this section. Let A_{spec} be a non-deterministic Büchi automaton that accepts the set of sequences in $(\{0,1\}^{N_I+N_O})^\omega$ that *do not* satisfy the specification. Let us also fix a set of Boolean variables B.

Consider the set of all trees corresponding to programs over the variables B with input and output arities N_I and N_O. By Lemma 1, this is a regular set of trees, and let's assume that A_{pgm} is a tree automaton that accepts precisely these trees.

We now build a two-way alternating Büchi tree automaton that accepts a tree encoding a program iff the program does *not* respect the specification. Intuitively, this automaton $\mathcal{A}$ will guess a particular run of the program by non-deterministically choosing a sequence of inputs, simulating the program on these inputs, and checking whether there is a run of the specification automaton $\mathcal{A}_{spec}$ that accepts this execution; if so, it will accept the tree.

The two-way alternating tree automaton will have as states two kinds of tuples. The first kind are tuples of the form $\langle s, q, i, m, t\rangle$ where s is the current state of the program's simulation (i.e., the valuation of the variables B), q is the current state of the specification automaton A_{spec} that is simultaneously simulated on the input-output sequence observed, i

is the *last* input received by the program, and m is a mode $m \in \{inp, out\}$ that remembers whether the next I/O operation the program must do is an input or an output. The final bit $t \in \{0,1\}$ is a toggle that switches to 1 each time the specification state is updated, and then gets set back to 0.

The second kind of state is of the form $\langle s, v\rangle$ where s is the current state of the program's simulation and v is a Boolean value; these states are used on subtrees that encode Boolean expressions (right-hand sides of assignments and conditionals in if- and while-statements), and are meant to check whether the expression evaluates to the value v in the current state of the program s.

Intuitively, the automaton walks over the program tree, interpreting every statement, and computing the current state of the variables in s. In this process, it may have to move up and down the tree as `while`-loops require traversing the same blocks of statements multiple times. When it meets an output-statement, it updates the specification automaton's state on the last input i and the output valuation. When it meets an input statement, it stores it in the appropriate variables in s, and updates the component i.

We now define the automaton formally. Let $Bool = \{0,1\}$. For such a valuation s of B, we denote by $s[b/v]$, where $b \in B$ and $v \in Bool$, the valuation that s' such that $s'(b) = v$ and for every $b' \in B, b' \neq b$, $s'(b') = s(b)$. We extend this to tuples of replacements: $s[\vec{b}/val]$ where val is a valuation of $\vec{b}$ is defined as the valuation s modified so that $\vec{b}$ evaluates to val.

Let S denote the set of all valuations of the variables B. Let $I = \{0,1\}^{N_I}$ denote the set of all inputs. Let $A_{spec} = (Q, q_0, \delta_{spec}, F_{spec})$.

The two-way alternating automaton is $\mathcal{A} = (P, p_0, \delta_L, \delta_R, \delta_{LR}, \delta_\emptyset, F)$ is defined as follows. The set of states is:

$$P = (S \times Bool) \cup (S \times Q \times I \times \{inp, out\} \times Bool)$$

The transitions are defined as follows, where $s \in S$, $v \in Bool$, $q \in Q$, $i \in I$, $m \in \{inp, out\}$, and $t \in \{0,1\}$.

- **Transitions from root:**
  $\delta_L(p_0, root, D) = (p_0, L)$; $\quad \delta_L((s,q,i,m,t), root, U) = true$;
- **Transitions to evaluate Boolean expressions:**
  - $\delta_\emptyset((s,1), \mathbf{tt}, D) = true$; $\quad \delta_\emptyset((s,0), \mathbf{tt}, D) = false$
  - $\delta_\emptyset((s,1), \mathbf{ff}, D) = false$; $\quad \delta_\emptyset((s,0), \mathbf{ff}, D) = true$
  - $\delta_\emptyset((s,v), b, D) = true \quad if\ s[b] = 1$
    $= false \quad otherwise.$
  - $\delta_{LR}((s,1), \vee, D) = ((s,1), L) \vee ((s,1), R)$
    $\delta_{LR}((s,0), \vee, D) = ((s,0), L) \wedge ((s,0), R)$
  - $\delta_L((s,v), \neg, D) = ((s, 1\text{–}v), L)$
- **Transitions to evaluate non I/O statements:**
  - $\delta_\emptyset((s,q,i,m,t), \mathbf{skip}, D) = ((s,q,i,m,0), U)$
  - $\delta_L((s,q,i,m,t), assign_b, D) =$
    $(\ (s[b/0], q, i, m, 0), U) \wedge ((s,0), L)\ )\ \vee\ (\ (s[b/1], q, i, m, 0), U) \wedge ((s,1), L)\ )$
  - $\delta_{LR}((s,q,i,m,t), \mathbf{if}, D) =$
    $(\ ((s,1), L) \wedge ((s,q,i,m,0), RL)\ )\ \vee\ (\ ((s,0), L) \wedge ((s,q,i,m,0), RR)\ )$
  - $\delta_{LR}((s,q,i,m,t), \mathbf{while}, D)$
    $= \delta_{LR}((s,q,i,m,t), \mathbf{while}, R)$
    $= (\ ((s,1), L) \wedge ((s,q,i,m,0), R)\ )\ \vee\ (\ ((s,0), L) \wedge ((s,q,i,m,0), U)\ )$
- **Transitions to evaluate input and output:**
  - $\delta_\emptyset((s,q,i,inp,t), \mathbf{input}\ \vec{b}) = \quad \bigvee_{valuations\ val\ over\ \vec{b}} ((s[\vec{b}/val], q, val, out, 0), U)$
  - $\delta_\emptyset((s,q,i,out,t), \mathbf{output}\ \vec{b}) = \quad \bigvee_{q' \in \delta_{A_{spec}}(q, \vec{i}, s[\vec{b}])} ((s, q', i, inp, 1), U)$

- **Transitions to move to next statement in program:**
  - $\delta_{LR}((s,q,i,m,t),\textbf{;},D) = ((s,q,i,m,t),L)$
  - $\delta_{LR}((s,q,i,m,t),\textbf{;},L) = ((s,q,i,m,t),R)$
  - $\delta_{LR}((s,q,i,m,t),\textbf{;},R) = ((s,q,i,m,t),U)$
  - $\delta_{LR}((s,q,i,m,t),\textbf{then},L)$
    $= \delta_{LR}((s,q,i,m,t),\textbf{then},R)$
    $= ((s,q,i,m,t),U)$
  - $\delta_{LR}((s,q,i,m,t),\textbf{if},R) = ((s,q,i,m,t),U)$

All other transitions evaluate to *false*.

The initial state is $p_0 = (s_0, q_0, i_0, \mathit{inp}, 0)$ where s_0 is the function that maps every variable in b to F (to reflect the initial state of the variables), q_0 is the initial state of the specification automaton A_{spec}, $i_0 = 0^{N_I}$ and where the mode is set to receive an input (which will overwrite i_0). This state is passed on from the root to the first statement of the program.

The set of Büchi final states is $F = \{(s,q,i,m,1) \mid s \in S, q \in F_{spec}, i \in I, m \in M\}$. Since the toggle t is 1 in all these states, the automaton is forced to truly hit a Büchi final state of A_{spec} infinitely often (as long as it runs forever), which in turn forces it to infinitely-often react with its environment.

A state of the form (s,v) is meant to check whether the state s satisfies the expression encoded at the subtree of the node the tree is reading. The transitions reflect this check: the expression b checks if it matches the value of v, and disjunctive and negated expressions are checked by sending appropriate copies to check sub-expressions. Note that these copies always go *down* the tree, and terminate at the leaves.

The processing of non-I/O statements requires walking up and down the tree. A skip-statement at a leaf is handled simply by going up. An assignment to a variable b is handled by guessing a Boolean value v, sending a copy down the tree to check that the expression evaluates to v, and sending another copy up with the value of b updated to v in the state. Conditionals are handled by again guessing whether the value of the expression is true or not, sending a copy to the left-child to check if this is correct, and sending a copy to the appropriate child of the right-tree to execute the if-branch or the else-branch. Recursive while-loops are handled similarly; the value of the condition is guessed, a copy is sent down the left-branch to check it, and another copy is either sent down to the right-branch to execute the body of the loop, or sent up to exit the loop.

The input-statement is handled only when in input mode (*inp*), and the automaton evaluates the input variables to an arbitrary valuation, stores this both in the state s and the input component i, and switches the mode to output (*out*). An output-statement does not change the state of the system, but changes the state of the specification, which is updated by simulating the specification automaton A_{spec} non-deterministically on the last input and the current output.

Notice that the last toggle bit t always gets set to 0, except when processing an output statement, where it gets set to 1. This ensures that specification states seen at intermediate points while simulating the program do not count towards meeting the Büchi specification in A_{spec}. Also note that if the program *halts* after a finite input sequence, the automaton will reach the root, and accept the tree; hence the automaton accepts also all programs that have a terminating computation.

There are also several other transitions that help move between statements. When a ";" is met going down a tree, the control goes to the left-child to process the first statement,

when it comes up from left-child, it goes to the right-child to process the second statement, and when that comes up, the control passes to the parent. When control comes up to an if- or then- node, the control moves up the tree.

We build a second two-way Büchi alternating automaton $A_{non\text{-}reactive}$ that accepts all programs that do *not* infinitely interact with the environment (i.e., those programs that never produce an output after a finite sequence of interactions with the environment). We skip this construction, as it is very similar and simpler than the construction of A above.

Finally, we take the union of the two two-way Büchi alternating tree-automata A and $A_{non\text{-}reactive}$, *complement* it (by dualizing the transition relation and making the acceptance condition co-Büchi), and intersect it with the automaton A_{pgm}. This gives a two-way alternating co-Büchi automaton that accepts precisely those programs that continually interact with the environment on all possible input sequences and satisfy the specification. This is then transformed to an equivalent non-deterministic tree-automaton, incurring an exponential blow-up.

The following theorem captures the correctness and complexity of the construction (note that $|B|$ dominates N_I and N_O); a gist of its proof can be found in [12].

▶ **Theorem 2.** *Let B be a finite set of Boolean variables, and let $N_I, N_O \in \mathbb{N}$. Let A_{spec} be a non-deterministic Büchi automaton over the alphabet $\{0,1\}^{N_I+N_O}$. Then we can construct a non-deterministic tree automaton $\mathcal{B}$ that precisely accepts the trees corresponding to reactive programs over B and with input/output type (N_I, N_O) that on all executions generate I/O sequences that are not in $L(A)$. Furthermore, this tree-automaton can be constructed to be of size $O(exp\ (|A_{spec}|,\ exp(B)))$.* ◀

Note that the final automaton is doubly-exponential in B and singly exponential in the size of the specification automaton. For a fixed B, N_I, N_O, this gives an ExpTime decision procedure. As a corollary, it follows that for specifications given in Ltl, the synthesis procedure is in 2ExpTime. Recall that the *transition-system* synthesis problem for LTL (with no parametrization like the variables B in program synthesis) is 2ExpTime-complete [17].

5 Synthesizing Recursive Reactive Programs

We extend the results of the previous section to synthesizing Boolean reactive programs *with function-calls and recursion*, provided the *number* of functions and their signatures are fixed. The proof strategy is similar but more complex: we encode programs with recursion also as trees, where functions are encoded as subtrees of the program tree, and show that the class of recursive programs that meet a regular specification is regular.

The idea of synthesizing recursive programs for regular specifications is motivated by three reasons. First, we are interested in synthesizing the smallest programs that satisfy a specification, and the smallest programs may involve recursion. Second, there may be *no programs* that satisfy a regular specification R over a set of Boolean variables B, while there *exist* recursive programs over B that satisfy R. This issue does not apply in the classical case of synthesizing transition systems as if there is a transition-system at all that satisfies a regular specification, there is always a finite-state transition system that satisfies it. In the program synthesis setting, the additional parametrization of B causes a smaller search space of programs. Finally, in applications to *program repair* (see Section 7 for a discussion), where we want to repair an existing recursive program against a regular specification, the natural problem that is required is one that synthesizes all recursive programs that satisfy a specification.

Our proof procedure also extends smoothly to synthesizing recursive programs against *visibly pushdown automata* [3] specifications, which are a class of specifications larger than that of regular specifications. This includes the class of temporal logics for recursive programs that can be compiled into visibly pushdown Büchi automata, such as CaRet [2] and and NWTL [1]. We note that we are unaware of any natural analog in the transition-system world corresponding to the results on synthesizing recursive programs against non-regular specifications in this section.

Recursive programs: Boolean programs with recursion are defined as sequential programs, except that we have functions that can call each other (with call-by-value semantics), functions have local variables that they can manipulate and can return a tuple of Boolean values, and can have side-effects by changing the valuation of globally declared variables.

Let us fix $B = (B_G, B_l)$, where B_G is a finite set of *global* Boolean variables and B_L and a finite set of local Boolean variables. Let us also fix a finite number of function-names F with a special function *main* in F. Let $Inp : F \to \mathbb{N}$ and $Ret : F \to \mathbb{N}$ be two functions that give the number of input parameters and the number of return values for each function, and let us fix $N_I, N_O \in \mathbb{N}$.

Then the set of recursive Boolean programs over $\langle B, F, In, Out, N_I, N_O\rangle$ is given by:

$$
\begin{array}{lcl}
\langle pgm\rangle & ::= & f(\vec{b})\ \{\ \langle stmt\rangle\ \} \ \mid\ \langle pgm\rangle\ \ \langle pgm\rangle \\
\langle stmt\rangle & ::= & \langle stmt\rangle; \langle stmt\rangle \ \mid\ \textbf{skip} \ \mid b := \langle expr\rangle \ \mid\ \textbf{input}\ \vec{b} \ \mid\ \textbf{output}\ \vec{b} \ \mid \\
 & & \textbf{if}\ (\langle expr\rangle)\ \textbf{then}\ \{\langle stmt\rangle\}\ \textbf{else}\ \{\langle stmt\rangle\} \ \mid\ \textbf{while}\ (\langle expr\rangle)\ \{\langle stmt\rangle\} \ \mid \\
 & & \langle b_1, \ldots, b_k\rangle = f(b'_1, \ldots, b'_i) \ \mid\ \textbf{return}\ (b_1, \ldots, b_r) \\
\langle expr\rangle & ::= & b \mid \textbf{tt} \mid \textbf{ff} \mid (\langle expr\rangle \vee \langle expr\rangle) \mid (\neg\langle expr\rangle)
\end{array}
$$

Programs consist of a series of definitions of the functions, where each function, in addition to the usual statements, is allowed to call other functions and assign the returned values to variables, as well as return a tuple of Boolean values. We assume natural restrictions on programs: each function is defined precisely once, and the arities of its input parameters and returns match the *In* and *Ret* functions.

The semantics is once again the natural one. When a function f_1 calls f_2, its local variables are pushed onto an (unbounded) stack along with the *program counter* at which the call occurred, and the control switches to the beginning of f_2, with its local variables reset to default initial values. When the function f_2 executes a **return**-statement, the stack is popped to retrieve the state of the caller function, the control switches to the caller at the appropriate program counter, and the returned values get stored in the appropriate variables mentioned in the statement that called f_2. The valuations of global variables can change across a call.

Representing recursive programs as finite trees: We can represent recursive programs using finite trees by essentially encoding each function as a sub-tree. Intuitively, we build a tree whose right-most path is labeled with a special symbol \$, and the sub-trees that hang from these nodes as left-children encode the various functions in the program.

Formally, we extend the function *tree* defined in the previous section with the following definitions, where $p, p' \in \langle pgm\rangle$.

$$
\begin{array}{rcl|rcl}
tree(p\ p') & = & \$(tree(p), tree(p')) & tree(\vec{b} := f(\vec{b'})) & = & call\text{-}f\text{-}\vec{b}\text{-}\vec{b'} \\
tree(f(\vec{b})\{\langle stmt\rangle\}) & = & f\text{–}\vec{b}(tree(\langle stmt\rangle)) & tree(\textbf{return}\ \vec{b}) & = & ret\text{-}\vec{b}
\end{array}
$$

Synthesizing recursive programs: The synthesis procedure is similar to that of non-recursive programs. We show that the class of *all* recursive programs over $\langle\langle B_G, B_L\rangle, F, In, Out, N_I, N_O\rangle$ that satisfy a specification (given by an automaton $\mathcal{A}_{spec}$ that accepts the sequences that do *not* confirm to the specification) is regular. The construction of the tree-automaton accepting the set of correct programs is considerably more complex. We give a gist of it below, highlighting the main parts of the construction.

Intuitively, when simulating the program, when we are at a state s, and the specification automaton state is q, and we process a *call* to a function f (i.e., a statement of the form $\vec{b} = f(\vec{b'})$), the tree automaton *guesses* the precise state s' and precise state q' the program and specification automaton would be in *after* returning from f, and sends two copies, one to continue computation in the current function from s' and q', and another to check whether the call to f does indeed transform the state from (s, q) to (s', q'). Note that the tree automaton is also guessing the *input* to the program on-the-fly as it simulates it; since the tree automaton is guessing only *one* input sequence on which the program responds, the methodology above to handle function calls works.

Recall the construction of the automaton in Section 4. The states there are of the form (s, q, i, m, t). In the new construction, the states will be of the form $(s, q, i, m, t, s', q', i', m', h)$ where $s, s' \in S$, $q, q' \in Q$, $m, m' \in \{inp, out\}$, and $t, h \in \{0, 1\}$. Intuitively, this state means that the program is in state (s, q, i, m, t) (as before) and is supposed to return from the current function at the state (s', q', m', t') (for some $t' \in \{0, 1\}$), and, if $h = 1$, it must see in the interim a Büchi final state.

The details of the construction are much more tedious, and we skip the construction here; a detailed construction can be found in [12]. The intent is however fairly straightforward: in a state $(s, q, i, m, t, \widehat{s}, \widehat{q}, \widehat{i}, \widehat{m}, h)$, reading a call to a function f (i.e., a statement of the form $\vec{b} := f(\vec{b'})$, the automaton will non-deterministically pick a quadruple (s', q', i', m') and $t', h', h'' \in \{0, 1\}$, and will (a) send a copy $(s'', q, i, m, s', q', i', m')$ up the tree to the first statement in the definition of the function f, where s'' is the appropriate state with formal parameters updated according to $\vec{b'}$, and will send another copy to the next statement in the current function in the state $(s''', q', i', m', 0, \widehat{s}, \widehat{q}, \widehat{i}, \widehat{m}, h'')$, where $h'' = h + h'$, and s''' is the state obtained from s by replacing $\vec{b}$ with the values returned at state s'. The latter copy will also have to pass through a transient intermediate Büchi final state if $h'' = 1$ (i.e., if f promises to meet a Büchi final state). Furthermore, we also need to provide a possibility for the function call f to *never return*; this will involve sending the current state to f with the proviso that if it meets a **return**-statement, then the tree would be rejected. We skip further details, and conclude with the main theorem for this section:

▶ **Theorem 3.** *Let $B = \langle B_G, B_L\rangle$ be a finite set of global and local Boolean variables, let F be a finite set of function-names, with arity functions In and Ret as above, and let $N_I, N_O \in \mathbb{N}$. Let A_{spec} be a non-deterministic Büchi automaton over the alphabet $\{0, 1\}^{N_I+N_O}$. Then we can construct a non-deterministic automaton that accepts precisely the trees corresponding to recursive reactive programs over $\langle B, F, In, Ret, N_I, N_O\rangle$ that on all executions generate I/O sequences that are not in $L(A)$. Furthermore, this tree-automaton can be constructed to be of size $O(exp\ (|A_{spec}|,\ |F|,\ exp(B)))$.* ◀

Handling visibly pushdown automata specifications:
The results in the above section smoothly extend to specification given as *non-deterministic visibly pushdown Büchi automata on ω-words.* [3].

Given a set of function-names F, a visibly pushdown automaton is over a triple alphabet $\langle\Sigma_c, \Sigma_r, \Sigma_i\rangle$ of calls, returns, and internal actions, respectively. A program's behavior is

redefined to be such that its function-calls and returns are made *visible*. More precisely, let us assume that $\Sigma_c = F$, $\Sigma_r = \{\overline{f} \mid f \in F\}$, and let $\Sigma_i = \{0,1\}^{N_I} \cup \{0,1\}^{N_O}$. Given a run of a program, we note down not just the input-output sequences it does (which have now been split), but also record the calls and returns the program makes. A visibly pushdown automaton is a pushdown automaton (a finite automaton with a stack) that is restricted so that it can *only* push on calls, only pop on returns, and cannot touch the stack on internal actions. Visibly pushdown automata can specify properties of the runs of a program: for example, given a pre- and post-condition for a function f, the visibly pushdown automaton can specify that f computes a function that conforms to it; such a specification is *not* regular but is a visibly pushdown language. Again, we assume that specifications are given by a visibly pushdown automaton that accepts the set of sequences that *do not* conform to the specification (visibly pushdown languages are closed under complement [3]).

Since the visibly pushdown automaton's stack is *synchronous* with the program's stack, the above synthesis procedure for recursive programs can be extended; at a call, the tree automaton will send an *updated* state of the specification (on the corresponding call) and update the automaton state on the corresponding return. We skip the construction, as it is almost identical to the the previous one save for the update of the specification automaton's state. The complexity of the construction also remains the same.

6 Discussion and Future Directions

While we have focused on imperative programs in this paper, the synthesized artefacts can be other compact representations in other domains as well: for example, synthesis problems can be targeted towards functional programs, towards non-reactive programs that compute one output from one input, or even hardware designs in a high-level hardware description language, like Verilog, RTL, or SystemC. We hope that the work presented here will inspire extending existing transition-system synthesis algorithms to artefact-oriented synthesis.

Program Repair: The results we have presented in this paper are extremely well-suited for *repairing programs*. Given a program P (with or without recursion) that does not satisfy a specification φ, the program-repair problem is to change P (in certain minimally allowed ways) so that it satisfies the specification φ.

Given a program P, let us assume that the set of *repaired versions* of the program that we want to search over is S. Repaired versions of programs may be defined as versions of the program obtained in certain ways, for example, changing only the conditionals in particular parts of the program, redefining only a particular function f, or having a hole in a program that needs to be filled by arbitrary code. Let us further assume that the trees corresponding to the programs in S defines a *regular* class of trees (this is a reasonable assumption; many repair conditions for programs are regular).

We can now synthesize a repair by constructing the class of all programs over the appropriate set of variables and functions (as defined by P and S), and construct a tree automaton $\mathcal{T}$ that accepts the trees of all these programs. The emptiness of the intersection of the languages of S and $\mathcal{T}$ gives the *repaired* versions of P that meet the specification.

A future direction we see is to *apply* Boolean program repair (facilitated by this paper) along with *abstraction* to repair programs over *unbounded* data domains.

Designing programs from automata accepting transition systems: One point worth making is that we can build programs from automata that accept transition systems. More precisely, assume that a transition-system synthesis procedure builds a tree-automaton $\mathcal{A}$ that accepts precisely the set of all trees that satisfy a particular specification. Then, we

can build a tree automaton $\mathcal{B}$ that accepts precisely the set of all trees that correspond to (non-recursive or recursive) programs whose transition-system unfolding is accepted by $\mathcal{A}$ (we skip the details here). Hence *any* synthesis procedure that builds a tree-automaton accepting unfoldings of transition systems can be turned into a synthesis procedure that constructs programs. However, the procedures laid out in the previous sections directly synthesize programs, and avoid the extra exponential blow-up that would be incurred by first building a tree automaton for transition systems followed by one for synthesizing programs.

Distributed synthesis: The above remark on synthesizing programs using a synthesizer of transition systems tempts us to think that any tree-automata based decision procedure for the synthesis problem for transition systems can be transformed to a synthesizer for programs. However, this is not clear for *distributed synthesis*, where we are required to synthesize programs at different sites of a distributed architecture with synchronous communication between sites [18].

First, the distributed transition-system synthesis problem is *undecidable* even in the simple architecture where there are two disconnected sites P_1 and P_2, each receiving inputs from the environment. It is not hard to adapt the undecidability proofs given in [18] to show that *program synthesis* for this architecture (as well as all other undecidable architectures for transition-system synthesis [18]) remain undecidable for program synthesis.

The classic *decidable* architecture for transition-system synthesis is that of a *pipeline* architecture, where the architecture consists of n processes, $P_1, \ldots, P_n$, where only P_1 receives input from its environment, where all processes have outputs, and where there are channels from P_i to P_{i+1}, for every $1 \leq i < n$. Pnueli and Rosner showed that this architecture has a decidable transition-system synthesis problem [18]. Their procedure (slightly modified) works by first taking the process P_1 and generating a tree-automaton accepting the set of all *communication trees over the first channel from P_1 to P_2* such that there is some strategy for P_1 to generate this tree and there is a strategy for the rest of the system (i.e., $P_2, \ldots P_n$) to generate outputs by reading this tree so as to satisfy the specification. The procedure then *walks* down the pipeline, producing at each point an automaton that accepts communication trees for the channels that admit a feasible synthesis. When we reach the last process, the procedure creates a transition system for P_n, and then *walks back* creating transition systems for the processes P_{n-1} all the way up to P_1.

The above decision procedure, however, does not seem adaptable for *program synthesis*. We can, of course, synthesize the program for P_n. But fixing a *particular program for P_n restricts the choices we have for other sites*. Consequently, when walking back, we may find that there is no program that satisfies the communication tree that we need for synthesis.

The distributed *program*-synthesis problem for pipelines is hence an open problem. Other decidable distributed synthesis problems, such as transition system synthesis for doubly-flanked pipelines against local specifications [14, 13]), also do not readily adapt to program synthesis, and remain open questions.

References

1 Rajeev Alur, Marcelo Arenas, Pablo Barceló, Kousha Etessami, Neil Immerman, and Leonid Libkin. First-order and temporal logics for nested words. *Logical Methods in Computer Science*, 4(4), 2008.

2 Rajeev Alur, Kousha Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *TACAS*, volume 2988 of LNCS, pages 467–481. Springer, 2004.

3 Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3), 2009.

4 Roderick Bloem, Stefan Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. Specify, compile, run: Hardware from PSL. *ENTCS*, 190(4):3–16, 2007.

5 J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. AMS*, 138:295–311, 1969.

6 Alonzo Church. Application of recursive arithmetic to the problem of circuit synthesis,. In *Summaries of talks presented at the Summer Institute for Symbolic Logic, Cornell University 1957*, pages 3–50, Princeton, 1957. Institute for Defense Analyses.

7 H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Draft, Available at `http://www.grappa.univ-lille3.fr/tata/`, 2002.

8 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of LNCS. Springer, 2002.

9 Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *POPL*, pages 317–330. ACM, 2011.

10 O. Kupferman and M.Y. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5(2):245 – 263, 1999.

11 Orna Kupferman, P. Madhusudan, P. S. Thiagarajan, and Moshe Y. Vardi. Open systems in reactive environments: Control and synthesis. In *CONCUR*, volume 1877 of LNCS, pages 92–107. Springer, 2000.

12 P. Madhusudan. *Synthesizing Reactive Programs*. Full version available at `http://www.cs.uiuc.edu/~madhu/csl11.html`.

13 P. Madhusudan. *Control and Synthesis of Open Reactive Systems*. PhD thesis, University of Madras, India, 2001.

14 P. Madhusudan and P. S. Thiagarajan. Distributed control and synthesis for local specificati ons. In *Proc., ICALP*, volume 2076 of *LNCS*, Greece, July 2001.

15 Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *VMCAI*, volume 3855 of LNCS, pages 364–380. Springer, 2006.

16 A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–77, 1977.

17 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM Symposium on Principles of Programming Languages*, 1989.

18 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *FOCS*, volume II, pages 746–757. IEEE, 1990.

19 M.O. Rabin. *Automata on infinite objects and Church's problem*. AMS, 1972.

20 S. Safra. On the complexity of ω-automata. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, pages 319–327, 1988.

21 Sven Schewe and Bernd Finkbeiner. Bounded synthesis. In *ATVA*, volume 4762 of LNCS, pages 474–488. Springer, 2007.

22 Armando Solar-Lezama. The sketching approach to program synthesis. In *APLAS*, volume 5904 of LNCS, pages 4–13. Springer, 2009.

23 Saurabh Srivastava, Sumit Gulwani, and Jeffrey S. Foster. From program verification to program synthesis. In *POPL*, pages 313–326. ACM, 2010.

24 W. Thomas. Languages, automata, and logic. *Handbook of Formal Language Theory*, III:389–455, 1997.

25 Wolfgang Thomas. Facets of synthesis: Revisiting church's problem. In *FOSSACS*, volume 5504 of LNCS, pages 1–14. Springer, 2009.

26 Moshe Y. Vardi. Reasoning about the past with two-way automata. In *ICALP*, volume 1443 of LNCS, pages 628–641. Springer, 1998.

27 M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

Concurrency Semantics for the Geiger-Paz-Pearl Axioms of Independence

Sara Miner More, Pavel Naumov, and Benjamin Sapp

Department of Mathematics and Computer Science
McDaniel College, Westminster, Maryland 21157, USA
{smore,pnaumov,brs004}@mcdaniel.edu

Abstract

Independence between two sets of random variables is a well-known relation in probability theory. Its origins trace back to Abraham de Moivre's work in the 18th century. The propositional theory of this relation was axiomatized by Geiger, Paz, and Pearl.

Sutherland introduced a relation in information flow theory that later became known as "non-deducibility." Subsequently, the first two authors generalized this relation from a relation between two arguments to a relation between two sets of arguments and proved that it is completely described by essentially the same axioms as independence in probability theory.

This paper considers a non-interference relation between two groups of concurrent processes sharing common resources. Two such groups are called non-interfering if, when executed concurrently, the only way for them to reach deadlock is for one of the groups to deadlock internally. The paper shows that a complete axiomatization of this relation is given by the same Geiger-Paz-Pearl axioms.

1998 ACM Subject Classification F.0 Theory of Computation

Keywords and phrases Independence, concurrency, information flow, axiomatization

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.443

1 Introduction

In this paper, we show that the same logical principles describe independence in three very different domains: probability, information flow, and concurrency.

1.1 Independence in Probability Theory

In probability theory, two events are called independent if the probability of their intersection is equal to the product of their probabilities. It is believed [6] that this notion was first introduced by de Moivre [2, 3]. If $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_m\}$ are two disjoint sets of random variables with finite ranges of values, then these two sets of variables are called independent if for any values $v_1, \dots, v_n$ and any values $w_1, \dots, w_m$, events $\bigwedge_{i\le n}(a_i = v_i)$ and $\bigwedge_{i\le m}(b_i = w_i)$ are independent. We denote this relation by $A \parallel B$. This definition can be generalized to independence of sets of variables with infinite ranges through the independence of appropriate σ-algebras.

A complete axiomatization of propositional properties of the independence relation between two sets of random variables was given by Geiger, Paz, and Pearl[1] [8]:

[1] The axiom names shown here are ours.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 443–457

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

1. Empty Set: $A \parallel \varnothing$,
2. Symmetry: $A \parallel B \to B \parallel A$,
3. Monotonicity: $A \parallel B, C \to A \parallel B$,
4. Exchange: $A, B \parallel C \to (A \parallel B \to A \parallel B, C)$,

where here and everywhere below A, B means the union of sets A and B. Furthermore, Studený [14] showed that *conditional* probabilistic independence does not have a complete finite axiomatization.

1.2 Independence in Information Flow

Sutherland [15] introduced a relation between two pieces of information, which we will call "secrets", that later became known as the "nondeducibility" relation. Two secrets are in this relation if any possible value of the first secret is consistent with any possible value of the second secret. More and Naumov [13] generalized this relation to a relation $A \parallel B$ between two sets of secrets and called it independence: sets of secrets A and B are independent if each possible combination of the values of secrets in A is consistent with each possible combination of the values of secrets in B. This relation also satisfies the Empty Set, Symmetry, Monotonicity, and Exchange axioms given above.

Describing the probabilistic semantics of relation $A \parallel B$, Geiger, Paz, and Pearl [8] assumed that sets A and B are disjoint since independence of a variable from itself is not a very intuitive idea. Under More and Naumov's semantics of secrets [13], however, $A \parallel A$ means that all secrets in set A have constant values which are known to everyone. More and Naumov called such secrets "public knowledge" and considered the relation $A \parallel B$ on sets of secrets where sets A and B are not necessary disjoint. They introduced a logical system that consists of the above Empty Set, Symmetry, Monotonicity, and Exchange axioms, as well as the following additional axiom:

5. Public Knowledge: $A \parallel A \to (B \parallel C \to A, B \parallel C)$.

They proved the completeness of this system with respect to a semantics of secrets. By analyzing their completeness proof, one can easily observe that if sets A and B are assumed to be disjoint, then the original four-axiom system of Geiger, Paz, and Pearl is complete with respect to the same semantics of secrets.

Cohen [1] presented a related notion called *strong dependence*. More recently, Halpern and O'Neill [9] introduced f-secrecy to reason about multiparty protocols. In our notation, f-secrecy is a version of the nondeducibility predicate whose left or right side contains a certain function of the secret rather than the secret itself. More and Naumov also axiomatized a variation of the independence relation between secrets over graphs [11, 5] and hypergraphs [12].

1.3 Independence in Concurrency Theory

In this paper, we propose a third semantics for the Geiger-Paz-Pearl axioms of independence. Under this semantics, independence is interpreted as "non-interference" between two sets of concurrent processes. Suppose that α and β are two such processes. We say that these processes interfere if they can deadlock when executed together. That is, there is a reachable state in which neither process can make a transition to another state, but at least one of the two processes can make a transition if the other process is not present.

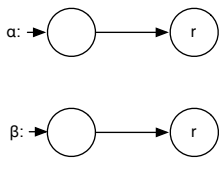


Figure 1 Two interfering processes.

One of the simplest examples of two such processes α and β is shown in Figure 1. Processes α and β both have initial states that require no resources and a second state in which the

process requires a resource r.

Suppose that process α makes a transition from the initial state to the second state. Then the whole system reaches deadlock although process β still would be able to make a transition in the absence of process α.

In this paper we will study the relation $A \parallel B$ between two *sets* of processes A and B. We will say that a set of processes A interferes with a set of processes B if these two sets can reach a deadlocked state where either set A or set B is not internally deadlocked.

For example, consider a variation of Dijkstra's dining philosopher problem depicted in Figure 2. It consists of three processes α, β, and γ, representing three dining philosophers. Each philosopher has access to two out of three resources r, s, and t, representing three forks in the dining philosophers problem. Each philosopher can acquire its two resources in any order, but needs both of them in order to "eat". Once a philosopher becomes full, he leaves the table and the process terminates.[2]

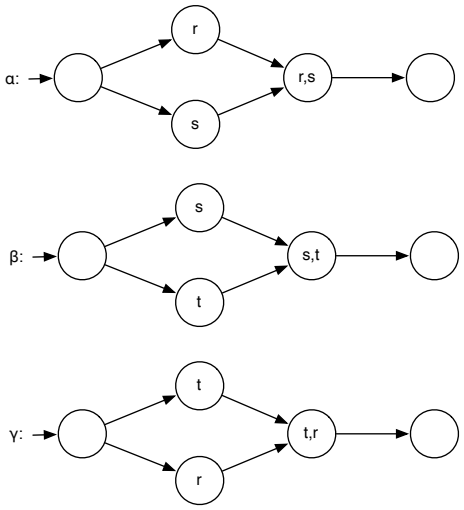


Figure 2 Three dining philosophers.

Let us first consider the concurrent execution of just two of these processes: α and β. Of course, if process α, for example, acquires resource s, then process β will need to wait until this resource is released before it will be able to finish. However, note that in any state of the composition of these two processes, at least one of the processes can make a transition, until both processes arrive at their respective final states. Thus, processes α and β do not interfere. We denote this non-interference by $\alpha \parallel \beta$.

The situation changes when all three processes are executed concurrently. If process α acquires resource r, process β acquires resource s, and process γ acquires resource t, then the system enters a deadlocked state in which none of the processes can make a transition. Yet, note that each process running alone can make a transition. In fact, any *pair* of processes running concurrently can make a transition in the absence of the third process. This means, for example, that the single process α interferes with the set of processes $\{\beta, \gamma\}$. In our notation, this can be expressed as $\neg(\alpha \parallel \beta, \gamma)$.

The main technical results of this paper are the soundness and completeness of the Geiger-Paz-Pearl logical system with respect to the non-interference semantics of concurrent processes sketched above. The significant implication of these results is that *the same non-trivial set of axioms captures the properties of independence in three very different settings: probability, information flow, and concurrency.*

2 Semantics

In order to prove formal results about process interference, we need to specify a mathematical model of concurrency. A number of models and formalisms for concurrent systems have been

[2] This is the form in which, with five philosophers rather than three, the problem was described by Hoare [10]. In Dijkstra's original version, "the life of a philosopher consists of an alternation of thinking and eating" ([4], p. 131), and, thus, graphs representing philosophers are cyclic.

developed. Among them are Petri nets, I/O automata, bigraphs, μ-calculus, and process calculi such as CCS, LOTOS, CADP, and Concurrency Workbench. (See, for example, Garavel [7], for a more recent review). Most of these were designed to be expressive enough to capture, at least potentially, reasoning about real-world systems. Since our ultimate goal is the completeness theorem, the *less expressive* our definition of concurrency, the stronger our result. Thus, instead of choosing one of the existing formalisms, we identified the minimal formalism sufficient for our proof of completeness. Specifically, we have chosen to define a process as a finite directed acyclic graph in which vertices are labeled by sets of resources. Figures 1 and 2 above depict examples of such processes. The concurrent execution of several such processes is captured in Definition 3 on page 447. We assume that concurrent processes defined using this formalism can also be captured, if needed, in other, richer, languages such as those mentioned above.

▶ **Definition 1.** A process is $\pi = (V, E, q, R, r)$, where

1. (V, E) is a finite directed acyclic graph (DAG). Vertices (elements of set V) will also be called "states" of the process.
2. $q \in V$ is a designated "initial" state of the process.
3. R is an arbitrary finite set of "resources" available to the process. Some of these resources may not actually be used by the process.
4. r is a "resource requirement" function from V to 2^R that specifies the resources used in each state. This function will be assumed to satisfy the following two conditions:
   a. $r(q) = \varnothing$,
   b. if $(v, w) \in E$, then $|r(w) \setminus r(v)| < 2$.

There are several aspects of our formalism that we would like to comment on.

Acquiring one resource at a time. Part 4 (b) of Definition 1 requires each process to acquire no more than one resource per transition. This is not a real restriction on the type of processes that we consider, but rather a restriction on how specific the description of a process should be. One always can introduce intermediate states in order to satisfy this requirement. For example, in Figure 3, instead of DAG α, one should specify DAG β, DAG γ, or DAG δ. This technical requirement is used in the proof of soundness of the Monotonicity axiom. Furthermore, in the conclusion, we will give an explicit example demonstrating that the Monotonicity axiom is false without this requirement.

Initial state. We assume that each process has a unique initial state. Additionally, we disallow processes which require resources in initial states so that each set of processes can be started concurrently. These are very technical limitations. If either condition is not satisfied, an artificial new initial state can always be added in order to satisfy it.

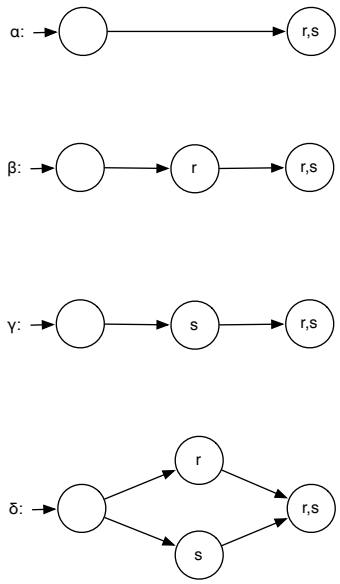


Figure 3 DAG α is not specific enough to be viewed as a process.

Resources at sink state. One might argue that, since all of our processes are finite DAGs, it is natural to assume that all processes must release all resources once they reach a sink state. We agree that this is a reasonable assumption to consider. However, our more general

approach will allow us to treat the concurrent execution of any set of several processes as a single process.[3]

Acyclic graphs. By representing a process as a finite DAG, we exclude from consideration any process that can run forever or that terminates after a number of steps which was unknown a priori. Considering such processes would create a whole new set of questions about fairness, livelock, etc. that would shift focus away from the deadlock interference that we consider in this paper. This certainly could be a direction for future work.

Resource multiplicity. Although our formalism does not allow for multiple copies of the same resource, one may still capture such processes by introducing distinct copies of these resources and additional states of the process for different combinations of them, as is done in the example in Figure 6 in the conclusion.

Resource production. Some models of concurrency, such as Petri nets, assume that processes not only "acquire" resources, but also "produce" new resources or additional copies of a resource not previously available in the system. Such processes are outside of the scope of this work, because the Monotonicity axiom does not hold for them.

▶ **Definition 2.** For any process $\pi = (V, E, q, R, r)$ and any state $v \in V$, we say that π is "alive" in v if there is $w \in V$ such that $(v, w) \in E$.

In other words, π is alive in v if v is not a sink of the directed acyclic graph (V, E). If π is alive in v, we will write $Alive_v(\pi)$.

For any $\pi = (V, E, q, R, r)$, by $State(\pi)$ we mean the set of all vertices V. By $State^R(\pi)$ we mean the set of all vertices of directed graph (V, E) reachable from the process' initial state q. By $Trans(\pi)$ we mean the set of transitions E. By $Res(\pi)$ we mean the set of resources R. By a family of processes $\{\pi_i\}_{i\in I}$ we mean any *multiset* of processes. That is, we allow some of the processes in the family to be equal.

The following is a key definition of this paper that formally captures the notion of concurrent execution of a family of processes.

▶ **Definition 3.** For any family of processes $\{\pi_i\}_{i\in I}$, such that $\pi_i = (V_i, E_i, q_i, R_i, r_i)$, the product of these processes $\prod_{i\in I} \pi_i$ is a tuple $\pi = (V, E, q, R, r)$, such that

1. V is a set of all tuples $\langle v_i\rangle_{i\in I} \in \prod_{i\in I} V_i$, where $r_i(v_i) \cap r_j(v_j) = \varnothing$ for all $i, j \in I$ such that $i \neq j$,
2. E is the set of all pairs $(\langle v_i\rangle_{i\in I}, \langle w_i\rangle_{i\in I}) \in V \times V$ such that there is $i_0 \in I$ for which $(v_{i_0}, w_{i_0}) \in E_{i_0}$ and $v_i = w_i$ for each $i \neq i_0$,
3. $q = \langle q_i\rangle_{i\in I}$,
4. $R = \bigcup_{i\in I} R_i$,
5. $r(\langle v_i\rangle_{i\in I}) = \bigcup_{i\in I} r(v_i)$.

Note the similarity between this definition and the Cartesian product of finite automata. A technical difference is in the fact that we disallow the simultaneous transitions of multiple processes. However, such simultaneous transitions can always be represented by a series of single transitions executed consecutively.

If set I is empty, then, as follows from the above definition, V consists of a single element – the 0-length tuple. We will denote this tuple by $\star$. The process which is the product of an empty family of processes will be denoted by ϵ. Thus, $\star \in State(\epsilon)$. If $I = \{i_1, \ldots, i_n\}$, then

[3] Even if the original processes release all resources in sink states, the concurrent execution of such processes may have sinks (deadlock states) in which some resources are not released.

we may informally denote $\prod_{i\in I}\pi_i$ by $\pi_{i_1}\times\cdots\times\pi_{i_n}$. However, since formally an element of $\prod_{i\in I}\pi_i$ is a function on I, the product is a commutative and associative operation.

▶ **Theorem 4.** *For any family of processes $\{\pi_i\}_{i\in I}$, the tuple $\prod_{i\in I}\pi_i$ is a process.* ◀

▶ **Definition 5.** A family of processes $\{\pi_i\}_{i\in I}$ is called "non-interfering" if for any $\langle v_i\rangle_{i\in I}\in State^R(\prod_{i\in I}\pi_i)$,

$$(\exists i\in I\ Alive_{v_i}(\pi_i))\to Alive_{\langle v_i\rangle_{i\in I}}\left(\prod_{i\in I}\pi_i\right).$$

▶ **Definition 6.** For any set I, the set of formulas $\Phi(I)$ is defined recursively: (i) $\bot\in\Phi(I)$, (ii) $(A\parallel B)\in\Phi(I)$, where A and B are two disjoint subsets of I, (iii) $\phi\to\psi\in\Phi(I)$, where $\phi,\psi\in\Phi(I)$.

▶ **Definition 7.** For any family of processes $\mathcal{P}=\{\pi_i\}_{i\in I}$ and any formula $\phi\in\Phi(I)$, we define the binary relation $\mathcal{P}\vDash\phi$ as follows:

1. $\mathcal{P}\nvDash\bot$,
2. $\mathcal{P}\vDash\phi\to\psi$ if and only if $\mathcal{P}\nvDash\phi$ or $\mathcal{P}\vDash\psi$,
3. $\mathcal{P}\vDash A\parallel B$ if and only if the two-element family of processes $\{\prod_{a\in A}\pi_a,\prod_{b\in B}\pi_b\}$ is non-interfering.

See Section 6.3 for a discussion of an n-ary version of the predicate $A\parallel B$.

3 Axioms

▶ **Definition 8.** The logic of concurrency, in addition to propositional tautologies and the Modus Ponens inference rule, consists of the following axioms:

1. Empty Set: $A\parallel\varnothing$,
2. Symmetry: $A\parallel B\to B\parallel A$,
3. Monotonicity: $A\parallel B,C\to A\parallel B$,
4. Exchange: $A,B\parallel C\to(A\parallel B\to A\parallel B,C)$.

We use notation $X\vdash\phi$ to denote that formula ϕ is provable in our logical system using the set of additional axioms X.

4 Soundness

The proof of soundness of the Geiger-Paz-Pearl axioms with respect to the non-interference semantics is not trivial. We state the soundness of each axiom as a separate theorem.

▶ **Theorem 9** (Empty Set). *No process α interferes with process ϵ.*

Proof. Consider any state $\langle a,\star\rangle\in State^R(\alpha\times\epsilon)$ such that $Alive_a(\alpha)$ or $Alive_\star(\epsilon)$. Note that $\star$ is the only state of process ϵ and, thus, $Alive_\star(\epsilon)$ is false. Hence, $Alive_a(\alpha)$. Thus, there is $a'\in State^R(\alpha)$ such that $(a,a')\in Trans(\alpha)$. Hence, $(\langle a,\star\rangle,\langle a',\star\rangle)\in Trans(\alpha\times\epsilon)$. Therefore, $Alive_{\langle a,\star\rangle}(\alpha\times\epsilon)$. ◀

▶ **Theorem 10** (Symmetry). *If process α does not interfere with process β, then process β does not interfere with process α.*

Proof. Consider any $\langle b,a\rangle\in State^R(\beta\times\alpha)$ such that $Alive_b(\beta)$ or $Alive_a(\alpha)$. By the assumption of the theorem, $Alive_{\langle a,b\rangle}(\alpha\times\beta)$. Since the product is a commutative operation, $Alive_{\langle b,a\rangle}(\beta\times\alpha)$. ◀

Next, we will prove the soundness of the Monotonicity axiom. It will be more convenient to prove the soundness of a slightly more general statement: $A, B \parallel C, D \to A \parallel C$.

▶ **Theorem 11** (Monotonicity). *For all processes $\alpha, \beta, \gamma, \delta$, if process $\alpha \times \beta$ does not interfere with process $\gamma \times \delta$, then process α does not interfere with process γ.*

Proof. Assume that process $\alpha \times \beta$ does not interfere with process $\gamma \times \delta$ and consider any state

$$\langle a, c\rangle \in State^R(\alpha \times \gamma) \tag{1}$$

such that $Alive_a(\alpha)$ or $Alive_c(\gamma)$. Without loss of generality, we will assume $Alive_a(\alpha)$. Thus, there is $a' \in State(\alpha)$ such that $(a, a') \in Trans(\alpha)$.

We need to show that $Alive_{\langle a,c\rangle}(\alpha \times \gamma)$. Indeed, assume that process $\alpha \times \gamma$ is deadlocked in state $\langle a, c\rangle$. Since $(a, a') \in Trans(\alpha)$, we must conclude that there is some resource $\rho_0 \in r(a) \setminus r(a')$ such that $\rho_0 \in r(c)$. By Definition 1, $|r(a') \setminus r(a)| < 2$ and, hence, $r(a') \setminus r(a) = \{\rho_0\}$.

Let b_0 and d_0 be the initial states of processes β and δ, respectively. Assumption (1) implies that $\langle a, b_0, c, d_0\rangle \in State^R(\alpha \times \beta \times \gamma \times \delta)$. Let process $\alpha \times \beta \times \gamma \times \delta$ transition from state $\langle a, b_0, c, d_0\rangle$ until it reaches a deadlock state $u \in State^R(\alpha \times \beta \times \gamma \times \delta)$. Since processes α and γ are themselves deadlocked in state $\langle a, c\rangle$, all transitions from $\langle a, b_0, c, d_0\rangle$ to u must have been made by processes β and δ. Thus, $u = \langle a, b, c, d\rangle$ for some $b \in State^R(\beta)$ and $d \in State^R(\delta)$. In other words,

$$\langle a, b, c, d\rangle \in State^R(\alpha \times \beta \times \gamma \times \delta)$$

and

$$\neg Alive_{\langle a,b,c,d\rangle}(\alpha \times \beta \times \gamma \times \delta). \tag{2}$$

The first of the above statements, by Definition 3, implies that $(r(a) \cup r(c)) \cap r(b) = \varnothing$. Recall that $r(a') \setminus r(a) = \{\rho_0\} \subseteq r(c)$. Thus,

$$r(a') \cap r(b) = ((r(a') \setminus r(a)) \cup (r(a') \cap r(a))) \cap r(b) \subseteq (r(c) \cup r(a)) \cap r(b) = \varnothing.$$

Hence $\langle a', b\rangle \in State(\alpha \times \beta)$. Recall that $(a, a') \in Trans(\alpha)$. Thus, $Alive_{\langle a,b\rangle}(\alpha \times \beta)$. By the assumption of the theorem, process $\alpha \times \beta$ does not interfere with process $\gamma \times \delta$. Hence, $Alive_{\langle\langle a,b\rangle,\langle c,d\rangle\rangle}((\alpha \times \beta) \times (\gamma \times \delta))$. Due to the associativity of the product operation, $Alive_{\langle a,b,c,d\rangle}(\alpha \times \beta \times \gamma \times \delta)$, which contradicts (2). ◀

▶ **Theorem 12** (Exchange). *For all processes α, β, γ, if process $\alpha \times \beta$ does not interfere with process γ and process α does not interfere with process β, then process α does not interfere with process $\beta \times \gamma$.*

Proof. Assume that process $\alpha \times \beta$ does not interfere with process γ and process α does not interfere with process β. Consider any $\langle a, \langle b, c\rangle\rangle \in State^R(\alpha \times (\beta \times \gamma))$. We need to prove that $Alive_{\langle a,\langle b,c\rangle\rangle}(\alpha \times (\beta \times \gamma))$ if either $Alive_a(\alpha)$ or $Alive_{\langle b,c\rangle}(\beta \times \gamma)$. Let us consider these two cases separately.

Case I. If $Alive_a(\alpha)$, then $Alive_{\langle a,b\rangle}(\alpha \times \beta)$, since process α does not interfere with process β. Thus, $Alive_{\langle\langle a,b\rangle,c\rangle}((\alpha \times \beta) \times \gamma))$, because process $\alpha \times \beta$ does not interfere with process γ. Therefore, since the product operation is associative, $Alive_{\langle a,\langle b,c\rangle\rangle}(\alpha \times (\beta \times \gamma))$.

Case II. If $Alive_{\langle b,c\rangle}(\beta \times \gamma)$, then there is $\langle b', c'\rangle \in State(\beta \times \gamma)$ such that $(\langle b, c\rangle, \langle b', c'\rangle) \in Trans(\beta \times \gamma)$ and either $b' = b$ or $c' = c$. Again, we need to consider two separate cases.

First, assume that $b' = b$. Hence, $Alive_c(\gamma)$. Thus, $Alive_{\langle\langle a,b\rangle,c\rangle}((\alpha \times \beta) \times \gamma))$, because process $\alpha \times \beta$ does not interfere with process γ. Therefore, since product is an associative operation, $Alive_{\langle a,\langle b,c\rangle\rangle}(\alpha \times (\beta \times \gamma))$.

Finally, suppose that $c' = c$. Hence, $Alive_b(\beta)$. Thus, $Alive_{\langle a,b\rangle}(\alpha \times \beta)$, because process α does not interfere with process β. Hence, $Alive_{\langle\langle a,b\rangle,c\rangle}((\alpha \times \beta) \times \gamma))$, because process $\alpha \times \beta$ does not interfere with process γ. Therefore, since product is an associative operation, $Alive_{\langle a,\langle b,c\rangle\rangle}(\alpha \times (\beta \times \gamma))$. ◀

5 Completeness

In this section we will prove the completeness of the Geiger-Paz-Pearl axioms with respect to non-interference semantics. This result is stated in Theorem 25. We start, however, with a sequence of lemmas in which we assume a fixed *finite* index set I and a fixed maximal consistent set of formulas $X \subseteq \Phi(I)$.

5.1 Critical Sets

▶ **Definition 13.** A set $C \subseteq I$ is called critical if there is a disjoint partition $C_1 \sqcup C_2$ of C, called a "critical partition", such that

1. $X \nvdash C_1 \parallel C_2$,
2. $X \vdash C_1 \cap D \parallel C_2 \cap D$, for any $D \subsetneq C$.

▶ **Lemma 14.** *Any critical partition is a non-trivial partition.*

Proof. It will be sufficient to prove that for any set A, we have $X \vdash A \parallel \varnothing$ and $X \vdash \varnothing \parallel A$. The first statement is an instance of Empty Set axiom, the second statement follows from Empty Set and Symmetry axioms. ◀

▶ **Lemma 15.** *$X \nvdash A \parallel B$, for any non-trivial (but not necessarily critical) partition $A \sqcup B$ of a critical set C.*

Proof. Suppose $X \vdash A \parallel B$ and let $C_1 \sqcup C_2$ be a critical partition of C. By the Monotonicity and Symmetry axioms, $X \vdash A \cap C \parallel B \cap C$. Thus,

$$X \vdash A \cap C_1, A \cap C_2 \parallel B \cap C_1, B \cap C_2. \tag{3}$$

Since $A \sqcup B$ is a non-trivial partition of C, sets A and B are both non-empty. Thus, $A \subsetneq C$ and $B \subsetneq C$. Hence, by the definition of a critical set, $X \vdash A \cap C_1 \parallel A \cap C_2$ and $X \vdash B \cap C_1 \parallel B \cap C_2$.

Note that $A \cap C$ is not empty since $A \sqcup B$ is a non-trivial partition of C. Thus, either $A \cap C_1$ or $A \cap C_2$ is not empty. Without loss of generality, assume that $A \cap C_1 \neq \varnothing$. From (3) and our earlier observation that $X \vdash A \cap C_1 \parallel A \cap C_2$, the Exchange axiom yields

$$X \vdash A \cap C_1 \parallel A \cap C_2, B \cap C_1, B \cap C_2.$$

By the Symmetry axiom,

$$X \vdash A \cap C_2, B \cap C_1, B \cap C_2 \parallel A \cap C_1. \tag{4}$$

The assumption $A \cap C_1 \neq \varnothing$ implies that $(A \cap C_2) \cup (B \cap C_1) \cup (B \cap C_2) \subsetneq C$. Hence, by the definition of a critical set, $X \vdash B \cap C_1 \parallel A \cap C_2, B \cap C_2$. By Symmetry axiom, $X \vdash A \cap C_2, B \cap C_2 \parallel B \cap C_1$. From (4) and the above statement, using the Exchange axiom,

$X \vdash A \cap C_2, B \cap C_2 \parallel A \cap C_1, B \cap C_1$. Since $A \sqcup B$ is a partition of C, we can conclude that $X \vdash C_2 \parallel C_1$. By the Symmetry axiom, $X \vdash C_1 \parallel C_2$, which contradicts the assumption that $C_1 \sqcup C_2$ is a critical partition.

◀

▶ **Lemma 16.** *For any two disjoint subsets $A, B \subseteq I$, if $X \nvdash A \parallel B$, then there is a critical partition $C_1 \sqcup C_2$, such that $C_1 \subseteq A$ and $C_2 \subseteq B$.*

Proof. Consider the partial order $\preceq$ on set $2^A \times 2^B$ such that $(E_1, E_2) \preceq (F_1, F_2)$ if and only if $E_1 \subseteq F_1$ and $E_2 \subseteq F_2$. Define $\mathcal{E} = \{(E_1, E_2) \in 2^A \times 2^B \mid X \nvdash E_1 \parallel E_2\}$. Note that $(A, B) \in \mathcal{E}$, because $X \nvdash A \parallel B$. Thus, $\mathcal{E}$ is a non-empty finite set. Take (C_1, C_2) to be a minimal element of set $\mathcal{E}$ with respect to partial order $\preceq$. ◀

5.2 Critical Set at a Dinner Table

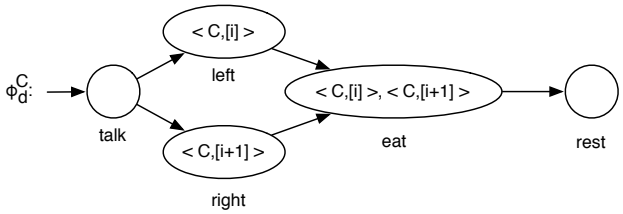


Figure 4 Critical set process ϕ_d^C.

For each critical set $C = \{c_1, \ldots, c_n\}$, we formally define the family of "dining philosophers" processes $\mathcal{P}_C = \{\phi_d^C\}_{d \in C} = \{(V, E, q, R^C, r_d^C)\}_{d \in C}$, shown in Figure 4, as follows

1. $V = \{talk, left, right, eat, rest\}$,
2. set E consists of edges $(talk, left)$, $(talk, right)$, $(left, eat)$, $(right, eat)$, $(eat, rest)$,
3. $q = talk$,
4. $R^C = \{C\} \times \mathbb{Z}_n$, or the set of all congruence classes in $\mathbb{Z}_n$ labeled with the critical set C. We will need this label later to distinguish resources of processes corresponding to different critical sets.
5. If $d = c_i$, then $r_d^C(talk) = r_d^C(rest) = \varnothing$, $r_d^C(left) = \{(C, [i])\}$, $r_d^C(right) = \{(C, [i+1])\}$, and $r_d^C(eat) = \{(C, [i]), (C, [i+1])\}$.

▶ **Lemma 17.** *For any $D \subseteq C$,*

$$\langle left \rangle_{d \in D} \in State^R \left(\prod_{d \in D} \phi_d^C \right).$$

Proof. Starting from the initial state $\langle talk \rangle_{d \in D}$, each of the processes $\{\phi_d^C\}_{d \in D}$ can make a transition into state $left$. ◀

▶ **Lemma 18.** *For any $D \subseteq C$,*

$$Alive_{\langle left \rangle_{d \in D}} \left(\prod_{d \in D} \phi_d^C \right) \quad \textit{iff} \quad D \neq C.$$

Proof. ($\Rightarrow$): Suppose $D = C$ and consider $\langle left\rangle_{c\in C}$, a state of process $\prod_{c\in C}\phi_c^C$. In this state no process can make a transition because all resources are already held.
($\Leftarrow$): Let $D \neq C$. Thus, there are more resources than processes. By the Pigeonhole Principle, if not all processes are in *rest* states, than at least one process has both of its resources available and, thus, can make a transition. ◀

▶ **Lemma 19.** *For any critical set C and any two disjoint subsets $A, B \subseteq C$, process $\prod_{a\in A}\phi_a^C$ and process $\prod_{b\in B}\phi_b^C$ interfere if and only if $A \sqcup B$ is a non-trivial partition of C.*

Proof. ($\Rightarrow$): Suppose that $A \sqcup B$ is not a non-trivial partition of C. Thus, either $A \cup B \subsetneq C$ or one of sets A and B is empty. In both of these cases, we need to prove that processes $\prod_{a\in A}\phi_a^C$ and $\prod_{b\in B}\phi_b^C$ do not interfere.
Case I. Suppose that $A \cup B \subsetneq C$. Consider any state

$$\langle s_a, s_b\rangle \in State^R\left(\prod_{a\in A}\phi_a^C \times \prod_{b\in B}\phi_b^C\right),$$

such that $Alive_{s_a}(\prod_{a\in A}\phi_a^C)$ or $Alive_{s_b}(\prod_{b\in B}\phi_b^C)$. Thus, $\langle s_a, s_b\rangle$ is not the state in which all ϕ-processes are already in state *rest*.

Since $A \cup B \subsetneq C$, there are more resources than ϕ-processes in the product $\prod_{a\in A}\phi_a^C \times \prod_{b\in B}\phi_b^C$. Thus, by the Pigeonhole Principle and since not all ϕ-processes are in *rest* states, at least one process has both of its resources available and, thus, can make a transition. Therefore,

$$Alive_{\langle s_a,s_b\rangle}\left(\prod_{a\in A}\phi_a^C \times \prod_{b\in B}\phi_b^C\right).$$

Case II. If one of sets A and B is empty, then the desired property follows from Theorem 9.
($\Leftarrow$): Consider state $\langle\langle left\rangle_{a\in A}, \langle left\rangle_{b\in B}\rangle$. By Lemma 17,

$$\langle\langle left\rangle_{a\in A}, \langle left\rangle_{b\in B}\rangle \in State^R\left(\prod_{a\in A}\phi_a^C \times \prod_{b\in B}\phi_b^C\right).$$

By Lemma 18, however,

$$Alive_{\langle left\rangle_{a\in A}}\left(\prod_{a\in A}\phi_a^C\right), \qquad Alive_{\langle left\rangle_{b\in B}}\left(\prod_{b\in B}\phi_b^C\right)$$

and

$$\neg Alive_{\langle\langle left\rangle_{a\in A}, \langle left\rangle_{b\in B}\rangle}\left(\prod_{a\in A}\phi_a^C \times \prod_{b\in B}\phi_b^C\right).$$

Therefore, processes $\prod_{a\in A}\phi_a^C$ and $\prod_{b\in B}\phi_b^C$ interfere. ◀

▶ **Lemma 20.** *For any two disjoint subsets $A, B \subseteq C$, if $X \vdash A \parallel B$, then processes $\prod_{a\in A}\phi_a^C$ and $\prod_{b\in B}\phi_b^C$ do not interfere.*

Proof. Suppose that processes $\prod_{a\in A}\phi_a^C$ and $\prod_{b\in B}\pi_b^C$ interfere. Thus, by Lemma 19, sets A and B form a non-trivial disjoint partition of set C. Hence, by Lemma 15, $X \nvdash A \parallel B$. ◀

▶ **Lemma 21.** *For any two families of processes $\{\alpha_j\}_{j\in J}$ and $\{\beta_j\}_{j\in J}$ such that sets $Res(\alpha_{j_1} \times \beta_{j_1})$ and $Res(\alpha_{j_2} \times \beta_{j_2})$ are disjoint for any $j_1 \neq j_2$, processes $\prod_{j\in J}\alpha_j$ and $\prod_{j\in J}\beta_j$ are non-interfering if and only if processes α_j and β_j are non-interfering for each $j \in J$.*

Proof. ($\Rightarrow$): Suppose that there is some j_0 such that processes α_{j_0} and β_{j_0} interfere. Thus, there is a state $\langle a, b\rangle \in State^R(\alpha_{j_0} \times \beta_{j_0})$ such that $\neg Alive_{\langle a,b\rangle}(\alpha_{j_0} \times \beta_{j_0})$ and either $Alive_a(\alpha_{j_0})$ or $Alive_b(\beta_{j_0})$. Without loss of generality, we will assume that $Alive_a(\alpha_{j_0})$.

Let $\langle q_j\rangle_{j\in J}$ be the initial state of process $\prod_{j\in J}(\alpha_j \times \beta_j)$. Define

$$q'_j = \begin{cases} \langle a, b\rangle & \text{if } j = j_0, \\ q_j & \text{otherwise.} \end{cases}$$

Since $\langle a, b\rangle \in State^R(\alpha_{j_0} \times \beta_{j_0})$, we can conclude that $\langle q'_j\rangle_{j\in J} \in State^R(\prod_{j\in J}(\alpha_j \times \beta_j))$. Let process $\prod_{j\in J}(\alpha_j \times \beta_j)$ start at state $\langle q'_j\rangle_{j\in J}$ and run until it reaches a state $\langle q''_j\rangle_{j\in J} \in State^R(\prod_{j\in J}(\alpha_j \times \beta_j))$ such that $\neg Alive_{\langle q''_j\rangle_{j\in J}}(\prod_{j\in J}(\alpha_j \times \beta_j))$. Let $q''_j = \langle a''_j, b''_j\rangle$. Because the product is a commutative and associative operation,

$$\neg Alive_{\langle\langle a''_j\rangle_{j\in J},\langle b''_j\rangle_{j\in J}\rangle}\left(\left(\prod_{j\in J}\alpha_j\right) \times \left(\prod_{j\in J}\beta_j\right)\right) \tag{5}$$

Since $\neg Alive_{\langle a,b\rangle}(\alpha_{j_0} \times \beta_{j_0})$, we can claim that $\langle a''_{j_0}, b''_{j_0}\rangle = q''_{j_0} = q'_{j_0} = \langle a, b\rangle$. Recall now our assumption that $Alive_a(\alpha_{j_0})$. Thus, $Alive_{a''_{j_0}}(\alpha_{j_0})$. Since, by the assumption of the lemma, any process α_j, where $j \neq j_0$, does not share resources with with process α_{j_0}, we can conclude $Alive_{\langle a''_j\rangle_{j\in J}}(\prod_{j\in J}\alpha_j)$. This, in conjunction with (5), implies that processes $\prod_{j\in J}\alpha_j$ and $\prod_{j\in J}\beta_j$ interfere.

($\Leftarrow$): Suppose that processes $\prod_{j\in J}\alpha_j$ and $\prod_{j\in J}\beta_j$ interfere. Thus, there is a state

$$\langle\langle a_j\rangle_{j\in J}, \langle b_i\rangle_{j\in J}\rangle \in State^R\left(\left(\prod_{j\in J}\alpha_j\right) \times \left(\prod_{j\in J}\beta_j\right)\right)$$

such that

$$\neg Alive_{\langle\langle a_j\rangle_{j\in J},\langle b_j\rangle_{j\in J}\rangle}\left(\left(\prod_{j\in J}\alpha_j\right) \times \left(\prod_{j\in J}\beta_j\right)\right) \tag{6}$$

but either $Alive_{\langle a_j\rangle_{j\in J}}(\prod_{j\in J}\alpha_j)$ or $Alive_{\langle b_j\rangle_{j\in J}}(\prod_{j\in J}\beta_j)$. Without loss of generality, assume that $Alive_{\langle a_j\rangle_{j\in J}}(\prod_{j\in J}\alpha_j)$. Thus, there is an $j_0 \in J$ such that $Alive_{a_{j_0}}(\alpha_{j_0})$. Hence, $Alive_{\langle a_{j_0},b_{j_0}\rangle}(\alpha_{j_0} \times \beta_{j_0})$, because, by the assumption of the lemma, processes α_{j_0} and β_{j_0} do not interfere. Thus, there is a state $\langle a', b'\rangle$ such that $(\langle a_{j_0}, b_{j_0}\rangle, \langle a', b'\rangle) \in Trans(\alpha_{j_0} \times \beta_{j_0})$. Since, by the assumption of the lemma, process $\alpha_{j_0} \times \beta_{j_0}$ does not share resources with any process $\alpha_j \times \beta_j$ such that $j \neq j_0$, the same transition is available to process $\prod_{j\in J}(\alpha_j \times \beta_j)$. Thus, $Alive_{\langle\langle a_j,b_j\rangle\rangle_{j\in J}}(\prod_{j\in J}(\alpha_j \times \beta_j))$. Due to the commutativity and associativity of the product, $Alive_{\langle\langle a_j\rangle_{j\in J},\langle b_j\rangle_{j\in J}\rangle}\left((\prod_{j\in J}\alpha_j) \times (\prod_{j\in J}\beta_j)\right)$, which contradicts (6). ◀

▶ **Definition 22.** $\mathcal{P} = \left\{\prod_{C\ni i}\phi_i^C\right\}_{i\in I}$, where the product is computed over all critical subsets C of I that contain i.

▶ **Lemma 23.** *For any disjoint subsets $A \subseteq I$ and $B \subseteq I$, $X \vdash A \parallel B$ if and only if $\mathcal{P} \vDash A \parallel B$.*

Proof. ($\Rightarrow$): Let $\mathcal{P} \nvDash A \parallel B$. Thus, processes $\prod_{a\in A}\prod_{C\ni a}\phi_a^C$ and $\prod_{b\in B}\prod_{C\ni b}\phi_b^C$ interfere. Hence, since the product operation is commutative and associative, processes $\prod_{C\in\mathcal{C}}\prod_{a\in A\cap C}\phi_a^C$ and $\prod_{C\in\mathcal{C}}\prod_{b\in B\cap C}\phi_b^C$ interfere, where $\mathcal{C}$ is the set of all critical subsets of set I.

For any two different critical sets C_1 and C_2, the set of resources available to process $\prod_{a\in A\cap C_1}\phi_a^{C_1}$ is $\cup_a R^{C_1} = \{C_1\}\times\mathbb{Z}_{|C_1|}$ and the set of resources available to process $\prod_{b\in B\cap C_2}\phi_b^{C_2}$ is $\cup_b R^{C_2} = \{C_2\}\times\mathbb{Z}_{|C_2|}$. These two sets of resources are disjoint since $C_1 \neq C_2$.

By Lemma 21, there must be a critical set $C_0 \in \mathcal{C}$ such that processes $\prod_{a\in A\cap C_0}\phi_a^{C_0}$ and $\prod_{b\in B\cap C_0}\phi_b^{C_0}$ interfere. Hence, by Lemma 20, $X \nvdash A\cap C_0 \parallel B\cap C_0$. By the Monotonicity axiom, $X \nvdash A\cap C_0 \parallel B$. By the Symmetry axiom, $X \nvdash B \parallel A\cap C_0$. Again by the Monotonicity axiom, $X \nvdash B \parallel A$. By the Symmetry axiom, $X \nvdash A \parallel B$, which is a contradiction.

($\Leftarrow$): Let $X \nvdash A \parallel B$. By Lemma 16, there is a critical set C such that $(A\cap C)\sqcup(B\cap C)$ is a critical partition of C. By Lemma 14, partition $(A\cap C)\sqcup(B\cap C)$ is a non-trivial partition of the critical set C. Thus, by Lemma 19, processes $\prod_{a\in A\cap C}\phi_a^C$ and $\prod_{b\in B\cap C}\phi_b^C$ interfere. By Lemma 21, processes $\prod_{C\in\mathcal{C}}\prod_{a\in A\cap C}\phi_a^C$ and $\prod_{C\in\mathcal{C}}\prod_{b\in B\cap C}\phi_b^C$ interfere. Since the product is a commutative and associative operation, processes $\prod_{a\in A}\prod_{C\ni a}\phi_a^C$ and $\prod_{b\in B}\prod_{C\ni b}\phi_b^C$ interfere. Therefore, $\mathcal{P} \nvDash A \parallel B$. ◀

▶ **Lemma 24.** *For any $\psi \in \Phi(I)$, $X \vdash \psi$ if and only if $\mathcal{P} \vDash \psi$.*

Proof. We use induction on structural complexity of formula ψ. The base case follows from Lemma 23, and the inductive case, after taking into account Definition 7, is straightforward. ◀

▶ **Theorem 25** (completeness)**.** *For any ϕ, if $\nvdash \phi$, then there is a family of processes $\mathcal{P} = \{\pi_i\}_{i\in I}$ such that $\mathcal{P} \nvDash \phi$.*

Proof. Assume that $\nvdash \phi$. Let I be the (finite) set of all indices used in formula ϕ and X be a maximal consistent subset of $\Phi(I)$ that contains formula $\neg\phi$. By Lemma 24, $\mathcal{P} \nvDash \phi$. ◀

6 Conclusions

6.1 The Monotonicity Axiom, Revisited

We will show that the Monotonicity axiom is not sound if the *acquire one resource at a time* condition is removed from Definition 1. Indeed, consider three "processes" specified by the DAGs in Figure 5. It will be sufficient to show that processes α and $\beta\times\gamma$ do not interfere, but processes α and β do interfere.

▶ **Theorem 26.** *Processes α and $\beta\times\gamma$ do not interfere.*

Proof. Consider any state $\langle a,\langle b,c\rangle\rangle \in State^R(\alpha\times(\beta\times\gamma))$ and assume that $\neg Alive_{\langle a,\langle b,c\rangle\rangle}(\alpha\times(\beta\times\gamma))$. We need to show that $\neg Alive_a(\alpha)$ and $\neg Alive_{\langle b,c\rangle}(\beta\times\gamma)$. Indeed, notice that the graph of process $\alpha\times(\beta\times\gamma)$ has only two sinks, thus the tuple $\langle a,\langle b,c\rangle\rangle$ has only two possible values.

Case I: $\langle a,\langle b,c\rangle\rangle = \langle 2,\langle 3,2\rangle\rangle$. Note that $\neg Alive_2(\alpha)$ and $\neg Alive_{\langle 3,2\rangle}(\beta\times\gamma)$.

Case 2: $\langle a,\langle b,c\rangle\rangle = \langle 2,\langle 1,2\rangle\rangle$. Note again that $\neg Alive_2(\alpha)$ and $\neg Alive_{\langle 1,2\rangle}(\beta\times\gamma)$. ◀

▶ **Theorem 27.** *Processes α and β interfere.*

Proof. Consider state $\langle 2,1\rangle \in State^R(\alpha\times\beta)$ and notice that $\neg Alive_{\langle 2,1\rangle}(\alpha\times\beta)$, but $Alive_1(\beta)$. ◀

6.2 The Public Knowledge Axiom

In Definition 6, we assumed that for $A \parallel B$ to be a valid formula, sets A and B must be disjoint. In the case of independence of secrets, More and Naumov [13] did not make this assumption. They noticed that $A \parallel A$ implies that each secret in set A has a constant and, thus, "publicly known" value. This led to an additional *Public Knowledge* axiom for independence:

$$A \parallel A \to (B \parallel C \to A, B \parallel C). \tag{7}$$

This Public Knowledge axiom, together with the Empty Set, Symmetry, Monotonicity, and Exchange axioms, forms a sound and complete system for the independence of secrets in information flow.

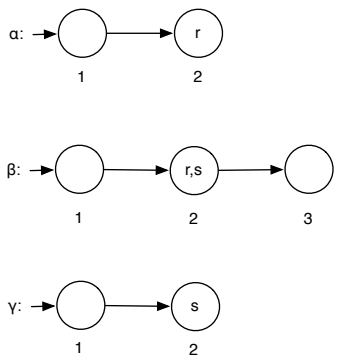


Figure 5 Monotonicity "counter-example."

Although Geiger, Paz, and Pearl [8] assumed that sets A and B were disjoint, this assumption is not important in their work. Indeed, it is easy to see that $A \parallel A$, under probability semantics, means that each variable in set A is constant almost everywhere. This means that the Public Knowledge axiom is also valid under the probability semantics. Finally, a review of the Geiger-Paz-Pearl completeness proof shows that a similar argument can be made in this more general setting if the Public Knowledge axiom is added to the system.

The case of concurrency semantics, however, is less straightforward. It depends on exactly what it means when the same process appears on both sides of $A \parallel B$. If $v \in A \cap B$, then one option is to assume that different occurrences of v refer to the same instance of a process. The other option is to assume that they refer to two different instances of the process. The former option requires them to have the same DAG and to be in the same states at any given time. The latter means that they have the same DAG, but could be in different states at any given time.

Under the first interpretation, $A \parallel A$ implies that each of processes in A cannot require any resources in reachable states, since otherwise both copies of the process would need to acquire the same resource. If the set of processes A does not require any resources in any of its reachable states, then it cannot affect interference between the other processes. Thus, the Public Knowledge axiom is sound. Moreover, the proof in this paper can be modified to show that the logical system formed by Empty Set, Symmetry, Monotonicity, Exchange, and Public Knowledge is complete under this interpretation.

Under the second interpretation, however, the Public Knowledge axiom is not sound. Indeed, consider the three processes α, β, and γ that have access to three resources r, s, and t. Each of the processes needs any two out of the three resources in order to terminate. Formally, all three of these processes have the same cube-shaped DAG, which is depicted in Figure 6. In some sense, this is a modified version of the Dining Philosopher's problem, where each of the philosophers α, β, and γ can use any two out of the three forks on the table.

Note that formula $\beta \parallel \gamma$ is true, because there are more resources than processes, thus, by the Pigeonhole Principle, at any state of $\beta \times \gamma$, either all processes have already terminated or one of the them has enough resources available to make a transition. For the same reason, formula $\alpha \parallel \alpha$ is also true as long as α on the left-hand-side and α on the right-hand-side refer to to two different instances of α.

Finally, formula $\alpha, \beta \parallel \gamma$ is false, because if processes α, β, and γ, respectively, acquire resources r, s, and t, then the system deadlocks. Therefore, the Public Knowledge axiom (7) is not sound if A, B, and C represent processes α, β, and γ.

6.3 An n-ary Non-interference Relation

In this paper, we considered the non-interference relation $A \parallel B$ between two sets of process. This binary relation can be generalized naturally to the n-ary relation $A_1 \parallel A_2 \parallel \cdots \parallel A_n$ between n sets of processes by changing part 4 of Definition 7 to

4. $\mathcal{P} \vDash A_1 \parallel A_2 \parallel \cdots \parallel A_n$ *if and only if the* n*-element family of processes* $\{\prod_{a \in A_i} \pi_a\}_{i \leq n}$ *is non-interfering.*

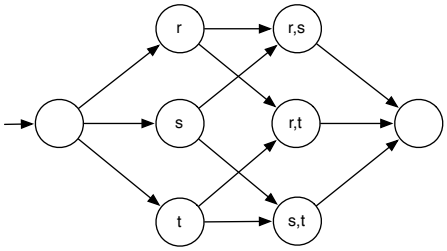


Figure 6 Cube DAG.

It turns out, however, that the n-ary non-interference relation can be expressed through the binary non-interference relation studied in this paper. For example, in the case where $n = 3$, the following result holds:

▶ **Theorem 28.** *For any family of processes* $\mathcal{P} = \{\pi_i\}_{i \in I}$ *and any subsets* A*,* B*, and* C *of set* I*,*

$$\mathcal{P} \vDash (A \parallel B \parallel C) \iff (A \parallel B, C) \wedge (B \parallel C).$$

Proof. In the following proof, we let α denote $\prod_{i \in A} \pi_i$, β denote $\prod_{i \in B} \pi_i$, and γ denote $\prod_{i \in C} \pi_i$.

($\Rightarrow$) : First, assume that $\mathcal{P} \nvDash A \parallel B, C$. Thus, there is a state $\langle a, b, c \rangle \in State^R(\alpha \times \beta \times \gamma)$ such that $\neg Alive_{\langle a,b,c \rangle}(\alpha \times \beta \times \gamma)$, but either $Alive_a(\alpha)$ or $Alive_{\langle b,c \rangle}(\beta \times \gamma)$. The latter implies that either $Alive_b(\beta)$ or $Alive_c(\gamma)$. Hence, we can conclude that at least one of the following three statements is true: $Alive_a(\alpha)$, $Alive_b(\beta)$, or $Alive_c(\gamma)$. By the assumption $\mathcal{P} \vDash A \parallel B \parallel C$, we can conclude that $Alive_{\langle a,b,c \rangle}(\alpha \times \beta \times \gamma)$, which is a contradiction.

Second, suppose that $\mathcal{P} \nvDash B \parallel C$. Thus, there is a state $\langle b, c \rangle \in State^R(\beta \times \gamma)$ such that $\neg Alive_{\langle b,c \rangle}(\beta \times \gamma)$, but either $Alive_b(\beta)$ or $Alive_c(\gamma)$. Let a_0 be the initial state of process α. Thus, $\langle a_0, b, c \rangle \in State^R(\alpha \times \beta \times \gamma)$. Let process $\alpha \times \beta \times \gamma$ make as many transitions as possible from state $\langle a_0, b, c \rangle$ until it reaches a state $\langle a', b', c' \rangle$ such that $\neg Alive_{\langle a',b',c' \rangle}(\alpha \times \beta \times \gamma)$. Note that $\neg Alive_{\langle b,c \rangle}(\beta \times \gamma)$ implies that $b' = b$ and $c' = c$. Thus, $\neg Alive_{\langle a',b,c \rangle}(\alpha \times \beta \times \gamma)$. However, we proved earlier that $Alive_b(\beta)$ or $Alive_c(\gamma)$. This contradicts our assumption that $\mathcal{P} \vDash A \parallel B \parallel C$.

($\Leftarrow$) : Let $\mathcal{P} \nvDash A \parallel B \parallel C$. Thus, there is a state $\langle a, b, c \rangle \in State^R(\alpha \times \beta \times \gamma)$ such that $\neg Alive_{\langle a,b,c \rangle}(\alpha \times \beta \times \gamma)$, but $Alive_a(\alpha)$, $Alive_b(\beta)$, or $Alive_c(\gamma)$.

If $Alive_a(\alpha)$, then, by the assumption $\mathcal{P} \vDash A \parallel B, C$, we can conclude that $Alive_{\langle a,b,c \rangle}(\alpha \times \beta \times \gamma)$, which is a contradiction.

If $Alive_b(\beta)$ or $Alive_c(\gamma)$, then $Alive_{\langle b,c \rangle}(\beta \times \gamma)$, by the assumption that $\mathcal{P} \vDash B \parallel C$. Thus, because $\mathcal{P} \vDash A \parallel B, C$, we have $Alive_{\langle a,b,c \rangle}(\alpha \times \beta \times \gamma)$, which is a contradiction. ◀

7 Acknowledgments

The authors would like to thank Joseph Halpern for pointing to the connection between the Geiger, Paz, and Pearl results on probabilistic independence and the first two authors' work on independence in information flow; and Sergei Artemov for the encouragement to look for new semantics of the axioms of independence.

References

1. Ellis Cohen. Information transmission in computational systems. In *Proceedings of Sixth ACM Symposium on Operating Systems Principles*, pages 113–139. Association for Computing Machinery, 1977.
2. Abraham de Moivre. De mensura sortis seu; de probabilitate eventuum in ludis a casu fortuito pendentibus. *Philosophical Transactions (1683-1775)*, 27:pp. 213–264, 1711.
3. Abraham de Moivre. *Doctrine of Chances*. 1718.
4. Edsger W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Inf.*, 1:115–138, 1971.
5. Michael Donders, Sara Miner More, and Pavel Naumov. Information flow on directed acyclic graphs. In L. Beklemishev and R. de Queiroz, editors, *Proceedings of 18th Workshop on Logic, Language, Information and Computation*. Springer, 2011. (to appear).
6. Moivre, Abraham, de. In *The New Encyclopædia Britannica*, volume 8, page 226. Encyclopædia Britannica, 15th edition, 1998.
7. Hubert Garavel. Reflections on the future of concurrency theory in general and process calculi in particular. *Electr. Notes Theor. Comput. Sci.*, 209:149–164, 2008.
8. Dan Geiger, Azaria Paz, and Judea Pearl. Axioms and algorithms for inferences involving probabilistic independence. *Inform. and Comput.*, 91(1):128–141, 1991.
9. Joseph Y. Halpern and Kevin R. O'Neill. Secrecy in multiagent systems. *ACM Trans. Inf. Syst. Secur.*, 12(1):1–47, 2008.
10. C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 26(1):100–106, 1983.
11. Sara Miner More and Pavel Naumov. On interdependence of secrets in collaboration networks. In *Proceedings of 12th Conference on Theoretical Aspects of Rationality and Knowledge (Stanford University, 2009)*, pages 208–217, 2009.
12. Sara Miner More and Pavel Naumov. Hypergraphs of multiparty secrets. In *11th International Workshop on Computational Logic in Multi-Agent Systems CLIMA XI (Lisbon, Portugal), LNAI 6245*, pages 15–32. Springer, 2010.
13. Sara Miner More and Pavel Naumov. An independence relation for sets of secrets. *Studia Logica*, 94(1):73–85, 2010.
14. Milan Studený. Conditional independence relations have no finite complete characterization. In *Information Theory, Statistical Decision Functions and Random Processes. Transactions of the 11th Prague Conference vol. B*, pages 377–396. Kluwer, 1990.
15. David Sutherland. A model of information. In *Proceedings of Ninth National Computer Security Conference*, pages 175–183, 1986.

Axiomatizing the Quote

Andrew Polonsky

VU University Amsterdam
De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands
andrew@few.vu.nl

Abstract

We study reflection in the Lambda Calculus from an axiomatic point of view. Specifically, we consider various properties that the quote $\ulcorner\cdot\urcorner$ must satisfy as a function from Λ to Λ. The most important of these is the existence of a definable left inverse: a term $\mathtt{E}$, called the *evaluator* for $\ulcorner\cdot\urcorner$, that satisfies $\mathtt{E}\ulcorner M\urcorner = M$ for all $M \in \Lambda$. Usually the quote $\ulcorner M\urcorner$ encodes the syntax of a given term, and the evaluator proceeds by analyzing the syntax and reifying all constructors by their actual meaning in the calculus. Working in Combinatory Logic, Raymond Smullyan [12] investigated which elements of the syntax must be accessible via the quote in order for an evaluator to exist. He asked three specific questions, to which we provide negative answers. On the positive side, we give a characterization of quotes which possess all of the desired properties, equivalently defined as being equitranslatable with a standard quote. As an application, we show that Scott's coding is not complete in this sense, but can be slightly modified to be such. This results in a minimal definition of a complete quoting for Combinatory Logic.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Lambda calculus, combinatory logic, quote operator, enumerator

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.458

1 Introduction

1.1 Coding in mathematical logic

Reflection is a powerful phenomenon in mathematical logic. Its most dramatic application was given by Gödel, who used it in the proof of his famous Incompleteness Theorems, destroying Hilbert's formalist program in its original incarnation (one could call the latter *Naïve Formalism*.) Soon after, it was at the heart of the proofs of equivalence between various models of computation that ultimately provided evidence for Church's thesis. Arithmetization of syntax is also the core component of the enumeration theorem, a result used implicitly in virtually every proof of Recursion Theory.

The ability of a computing system to interpret its own syntax also played a significant role in the evolution of functional programming languages. In one of the early reports on the development of Lisp, John McCarthy [8] introduced the so-called Meta-Circular Evaluator: a Lisp form which can execute an arbitrary list as a Lisp form — a "universal Lisp form." Since then, many languages (including Lisp, Prolog, Smalltalk, and others) have been built ground-up using meta-circular implementation. In the reverse direction, some languages have the "quote" command, which represents expressions of the language within some standard datatype. This operation is not referentially transparent, so the presence of an explicit quote operator in a language (e.g. Lisp) means that the language is not purely functional. Nevertheless, computational reflection provides the language with other powerful capabilities, which were extensively investigated by Brian Cantwell Smith in his PhD thesis [11].

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 458–469

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

The peculiar use of self-reference made Gödel's argument a favorite among philosophers, and inspired a number of publications in popular science, some of which even attribute a certain mystical element to the work of Gödel. For example, in his introduction to *Gödel, Escher, Bach: an Eternal Golden Braid*, Hofstadter writes: "GEB is in essence a long proposal of strange loops as a metaphor for how selfhood originates."[6] Although Hofstadter's allegorical picture cannot be framed as a scientific thesis, it did stimulate popular interest in computational logic.

The questions of Smullyan were brought to our attention by Henk Barendregt. Of course, they are only a sliver in the more global puzzle of understanding reflection as a distinct phenomenon. There is still lacking a general concept, an all-inclusive definition through which the common features of the constructions in Gödel's theorem, computability, number theory (systems of arithmetic), and set theory could be related. Finding such a concept remains a fascinating open problem.

1.2 Coding of lambda terms

Classically, an *enumerator* is a term $\mathtt{E}$ such that every closed lambda term is convertible[1] to $\mathtt{E}\mathtt{c}_n$ for some natural number n, where $\mathtt{c}_n$ denotes the nth Church numeral. The first enumerator for the lambda calculus was constructed by Kleene [7] in the proof that every lambda-definable function is computable — among the first pieces of evidence for the Church–Turing thesis. Together with the proof that every computable function is lambda-definable, this gave an interpretation of lambda-calculus within itself. Kleene's approach used Gödel's arithmetization of syntax, which codes grammar trees of terms as natural numbers. This has the drawback that an evaluator exists only for terms whose free variables come from a finite set which is fixed in advance.

Mogensen [9] found an elegant self-interpreter which, instead of coding variables by numerals, coded them by themselves. The coding therefore allows an evaluator which is uniform on the set of all (open) lambda terms. Mogensen's construction has a different drawback: it lacks a *discriminator* — a term which can test whether or not two quotes code the same term. However, Barendregt [2] did find a discriminator for Mogensen coding which works for all closed terms.

A more significant distinction between Kleene's enumerator and Mogensen's is that Kleene actually emulates variable binding within the quotes. This requires a number of auxiliary functions to deal with alpha-conversion, making definitions rather complicated. In contrast, Mogensen encodes binders by actual "meta-level" lambdas. This technique is known as Higher Order Abstract Syntax [10], and Mogensen's coding is arguably the most canonical application of it.

In 1992, Berarducci and Böhm gave an improvement on Mogensen's coding such that the evaluator $\mathtt{E}$ is a normal form and $\mathtt{E}\ulcorner M\urcorner$ is strongly normalizing whenever M is. They also listed other properties that a coding might satisfy, and reiterated the problem of axiomatizing the quote as an operator. [5] Our proposed solution appears in Corollary 14.

To keep matters simple, we will restrict attention to the coding of *closed* terms, and work in the combinatory version of the lambda calculus with basis $\{\mathtt{K}, \mathtt{S}\}$. This results in no loss of generality, as all closed lambda terms can always be written in this basis. Indeed,

[1] In fact, in the lambda caluclus all enumerators are actually *reducing*: if $\mathtt{E}$ is an enumerator, then $\forall M \in \Lambda^0\ \exists n \in \mathbb{N}$ s.t. $\mathtt{E}\mathtt{c}_n \twoheadrightarrow M$. Richard Statman gave the first proof of this result using computability theory, and Henk Barendregt provided a constructive adaptation, which can be found in the festschrift of Dirk van Dalen [3].

our constructions can be translated into Mogensen coding rather explicitly. Furthermore, as will be evident from the definitions, the choice of basis has no effect on our results.

2 Setup

2.1 Basic concepts

In what follows, we will need the following concepts. For a thorough introduction, see [1].

▶ **Definition 1.** Let $V = \{v_0, v_1, \dots\}$ be an infinite set of variables.

1. The *lambda terms* are given by the grammar

$$\Lambda ::= V \mid \Lambda\Lambda \mid \lambda V\Lambda$$

2. A subterm occurrence of a variable x in the term M is *bound* if it is inside a subterm of the form (λxN). Otherwise, the occurrence is *free*. $\mathsf{FV}(M)$ denotes the set of variables that have a free occurrence in M. If $\mathsf{FV}(M) = \emptyset$, then M is *closed*, and we write $M \in \Lambda^0$.
3. $M[x := N]$ is the lambda term obtained by renaming bound variables of M to be distinct from the free variables of N, and then plugging in the term N for every free occurrence of x in the resulting M.
4. Lambda terms are considered for their relation of *beta-convertibility* — a congruence generated by the axiom

$$(\lambda xM)N =_\beta M[x := N]$$

5. As a matter of notation, we write
   - $M_1M_2 \dots M_n$ as a shorthand for $(\dots(M_1M_2)M_3)\dots)M_n$,
   - $\lambda\vec{x}.M$ as a shorthand for $\lambda x_0(\lambda x_1 \dots (\lambda x_l M)\dots)$.
6. The *combinators* are given by the grammar

$$\mathcal{C} ::= \mathtt{K} \mid \mathtt{S} \mid \mathcal{C}\mathcal{C}$$

   The combinator $\mathtt{SKK}$ is abbreviated by the symbol $\mathtt{I}$.
7. The combinators are considered with the congruence generated by equations
   - $\mathtt{K}xy = x$
   - $\mathtt{S}xyz = (xz)(yz)$
8. Lambda terms are translated into combinators via the map $(\cdot)_{CL} : \Lambda \to \mathcal{C}$:
   - $(x)_{CL} = x$
   - $(MN)_{CL} = (M)_{CL}(N)_{CL}$
   - $(\lambda x.M)_{CL} = \lambda^* x[(M)_{CL}]$,

   where $\lambda^* x[\cdot]$ is given by
   - $\lambda^* x[x] = \mathtt{I}$
   - $\lambda^* x[M] = \mathtt{K}M$, if $x \notin \mathsf{FV}(M)$
   - $\lambda^* x[MN] = \mathtt{S}(\lambda^* x[M])(\lambda^* x[N])$

   Note that when M is closed, $(M)_{CL}$ has no variables (i.e., $(M)_{CL} \in \mathcal{C}$).
9. The basic combinators are represented in Λ by the terms

$$\mathtt{I} = \lambda x.x, \quad \mathtt{K} = \lambda xy.x, \quad \mathtt{S} = \lambda xyz.xz(yz)$$

   In addition, we'll employ the following standard abbreviations:
   - $\mathtt{U}_i^n = \lambda x_0 \dots x_n.x_i$
   - $\overline{\mathtt{K}} = \mathtt{U}_1^1 = \lambda xy.y$

- $\mathtt{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$
- $\Omega = (\lambda x.xx)(\lambda x.xx)$
- $[M,N] = \lambda x.xMN, \qquad x \notin \mathsf{FV}(MN)$
- $\langle M_1,\dots,M_n\rangle = \lambda x.xM_1\dots M_n, \qquad x \notin \mathsf{FV}(M_1)\cup\dots\cup\mathsf{FV}(M_n)$

▶ **Remark 2.** We will often mix together lambda terms and combinators, leaving the translation above implicit in notation. Since we work in combinatory logic, this means that all occurrences of λ are to be eliminated via part 8 of the definition above.

In what follows, we will need to have a standard, reference coding with which others can be compared. Any of those mentioned previously would work; our variant uses pairing to represent the syntax trees.

▶ **Definition 3.** The *standard quote* of M is defined inductively as follows. Let $\mathtt{P} \equiv (\lambda xyz.zxy)_{CL}$ be the pairing combinator. Put

- $\underline{M} \equiv \mathtt{SI}(\mathtt{K}M) = \langle M\rangle \qquad$ if $M \in \{\mathtt{K},\mathtt{S}\}$,
- $\underline{MN} \equiv \mathtt{P}\,\underline{M}\,\underline{N} = [\underline{M},\underline{N}] \qquad$ for all M,N.

2.2 Axioms for the quote operator

A *coding* $\ulcorner\cdot\urcorner$ is a map from $\mathcal{C}$ into itself. A term $\ulcorner M\urcorner$ is then called *the quote of* M.[2] Since the primary use of coding consists of manipulating the syntax of terms, most of the properties that we investigate will concern existence of combinators relating the structure of a term to that of its quote. Among these, most attention is given to the Constructor and Destructor axioms. Roughly, the former allows one to obtain the quote of a term from the quotes of its subterms. The latter is dual: it breaks up the term into its subterms (with respect to the quote).

▶ **Definition 4.** (Coding Axioms) Let $\ulcorner\cdot\urcorner : \mathcal{C}\to\mathcal{C}$. We say $\ulcorner\cdot\urcorner$ *satisfies axiom* X from among those below if there exists a combinator X with the stated property.

CON (*constructor*) :
$$\begin{cases} \text{A:} & A\ulcorner M\urcorner\ulcorner N\urcorner = \ulcorner MN\urcorner \\ \text{B:} & B\ulcorner M\urcorner = \ulcorner\ulcorner M\urcorner\urcorner \end{cases}$$

DES (*destructor*) :
$$\begin{cases} \text{P:} & P_i\ulcorner M_0M_1\urcorner = \ulcorner M_i\urcorner, \quad i\in\{0,1\} \\ \text{Z:} & Z_b\ulcorner M\urcorner = \begin{cases} \mathtt{K} & M\equiv b \\ \overline{\mathtt{K}} & \text{otherwise} \end{cases} \qquad b\in\{\mathtt{K},\mathtt{S}\} \end{cases}$$

CMP (*complete*) :
$$\begin{cases} \text{U:} & U\ulcorner M\urcorner = \underline{M} & \text{(uncoding)} \\ \text{U}^{-1}\text{:} & U^{-1}\underline{M} = \ulcorner M\urcorner & \text{(encoding)} \end{cases}$$

E (*evaluator*) : $\mathtt{E}\ulcorner M\urcorner = M$

Δ (*discriminator*) : $\Delta\ulcorner M\urcorner\ulcorner N\urcorner = \begin{cases} \mathtt{K} & M\equiv N \\ \overline{\mathtt{K}} & \text{otherwise} \end{cases}$

MON (*monic*) : $\forall M,N\in\mathcal{C} \qquad \ulcorner M\urcorner = \ulcorner N\urcorner \Longrightarrow M\equiv N$

SOL (*solvable*) : $\forall M\in\mathcal{C} \qquad \ulcorner M\urcorner$ is solvable

[2] Some authors would call $\ulcorner M\urcorner$ a *quasiquote*, but we will not make this distinction here.

▶ **Remark 5.** Smullyan called a coding satisfying CON *admissible*, and a coding satisfying DES *preadmissible* [12, p.367]. He asked whether either implies the other, and whether an evaluator can be constructed from CON. All three questions have negative answers.

▶ **Remark 6.** Axiom B appears to be too strong: if we want to requote M, why should we care about *the particular* $=_\beta$*-representative* of $\ulcorner M \urcorner$? It may be more reasonable to require

$$\mathsf{B}^-\text{:} \qquad B^- \ulcorner M \urcorner = \ulcorner N \urcorner, \text{ where } N = \ulcorner M \urcorner$$

Nevertheless, we will proceed with Smullyan's original formulation.

The axioms above are the primary focus of our attention. In studying them, the following auxiliary properties are useful.

▶ **Definition 7.** We introduce two additional axioms

$$\mathrm{Z}_?\ (\textit{leaf test}): \qquad Z_? \ulcorner M \urcorner = \begin{cases} \mathtt{K} & M \in \{\mathtt{K}, \mathtt{S}\} \\ \overline{\mathtt{K}} & M \equiv M_0 M_1 \end{cases}$$

$$\mathrm{R}_D\ (\textit{range test}): \quad \exists\ c.e.\ D \subseteq \mathcal{C},\ \mathsf{Range}(\ulcorner \cdot \urcorner) \subseteq D,\ \exists R_D \in \mathcal{C},\ \forall N \in D:$$

$$R_D N = \begin{cases} \mathtt{K} & \exists M.\ N = \ulcorner M \urcorner \\ \overline{\mathtt{K}} & \text{otherwise} \end{cases}$$

3 Results

3.1 Elementary properties

▶ **Proposition 8.** *Let* $\ulcorner \cdot \urcorner$ *be a coding. Then the following implications hold:*

1. $\ulcorner \cdot \urcorner \equiv \mathtt{.} \implies \mathrm{CON} \wedge \mathrm{DES} \wedge \mathrm{E} \wedge \Delta$
2. $\mathrm{Z} \implies \mathrm{SOL}, \quad \mathrm{DES} \implies \mathrm{MON}, \quad \Delta \implies \mathrm{MON} \wedge \mathrm{Z}$
3. $\mathrm{CMP} \implies \mathrm{CON} \wedge \mathrm{DES} \wedge \Delta$
4. $\mathrm{DES} \implies \mathrm{U}, \quad \mathrm{U} \implies \mathrm{E} \wedge \Delta$

Proof. 1. We verify that the standard coding has all of the properties of interest.

- Let $P_i = \langle \mathtt{U}_i^1 \rangle_{CL} = \mathtt{SI}(\mathtt{KU}_i^1)$. Then

$$\begin{aligned} P_i[M_0, M_1] &= \mathtt{I}[M_0, M_1](\mathtt{KU}_i^1[M_0, M_1]) \\ &= (\lambda x_0 x_1 . x_i) M_0 M_1 = M_i \end{aligned}$$

  In particular, $P_i \underline{M_0 M_1} = \underline{M_i}$.
- Let $Z_? = (\lambda x . x \mathtt{U}_2^3 \mathtt{I})_{CL}$. (From now on, $(-)_{CL}$ will be left implicit as per Remark 2.) Then

$$\begin{aligned} Z_? \langle x \rangle &= \mathtt{K} \\ Z_? [x, y] &= \overline{\mathtt{K}} \end{aligned}$$

  In particular, $Z_? \underline{MN} = Z_? [\underline{M}, \underline{N}] = \overline{\mathtt{K}}$, and $Z_? \underline{\mathtt{K}} = Z_? \underline{\mathtt{S}} = \mathtt{K}$.
- With $\mathrm{Z}_?$ satisfied, it is trivial to get full Z. Since $\mathtt{K}, \mathtt{S}$ are normal forms, by Böhm's theorem [1], there exist closed terms $\overrightarrow{Q}$ such that $\underline{\mathtt{K}} \overrightarrow{Q} = \mathtt{K}$ and $\underline{\mathtt{S}} \overrightarrow{Q} = \overline{\mathtt{K}}$. Take

$$\begin{aligned} Z_{\mathtt{K}} x &= \text{IF } Z_? x \text{ THEN } x \overrightarrow{Q} \text{ ELSE } \overline{\mathtt{K}} \\ Z_{\mathtt{S}} x &= \text{IF } Z_? x \text{ THEN } x \overrightarrow{Q} \overline{\mathtt{K}} \mathtt{K} \text{ ELSE } \overline{\mathtt{K}} \end{aligned}$$

- So far we have proved that $\cdot$ satisfies DES. To satisfy axiom A, simply take $A = \mathtt{P}$. Furthermore, this is the representative of the β-equivalence class of $\underline{MN}$ that was chosen by Definition 3: $\underline{MN} \equiv (\mathtt{P}\,\underline{M})\,\underline{N}$.
- Using a fixed-point combinator, put

$$Bx = \text{IF } Z_?x \text{ THEN } (Z_{\mathtt{K}}x)\underline{\mathtt{K}}\,\underline{\mathtt{S}} \text{ ELSE } \mathtt{P}(\mathtt{P}\underline{\mathtt{P}}(B(P_0x)))(B(P_1x))$$

- Evaluator is easy for the standard coding:

$$\mathtt{E}x = \text{IF } Z_?x \text{ THEN } x\mathtt{I} \text{ ELSE } x(\lambda xy.\mathtt{E}x(\mathtt{E}y))$$

- So is the discriminator:

$$\begin{aligned}\Delta xy =&\text{IF } Z_?x\\ &\text{THEN IF } Z_{\mathtt{K}}x \text{ THEN } Z_{\mathtt{K}}y \text{ ELSE } Z_{\mathtt{S}}y\\ &\text{ELSE IF } Z_?y \text{ THEN } \overline{\mathtt{K}} \text{ ELSE } (\Delta(P_0x)(P_0y))(\Delta(P_1x)(P_1y))\overline{\mathtt{K}}\end{aligned}$$

2. That Z $\Longrightarrow$ SOL is an immediate consequence of the Genericity Lemma [1, 14.3.24]: if $ZM = \mathtt{K}$ for unsolvable M, then $Zm = \mathtt{K}$ for all M, contradicting condition Z.
   That $\Delta \Longrightarrow$ MON $\wedge$ Z is also immediate.
   To see that DES $\Longrightarrow$ MON we proceed by induction on the height of M. The base step is assured by Z. If $M \equiv M_0M_1$, and $N \equiv N_0N_1$, then $(M \not\equiv N) \Longrightarrow (M_i \not\equiv N_i)$ for some i. If $\ulcorner M \urcorner = \ulcorner N \urcorner$, then applying the i'th projection contradicts the inductive hypothesis.
3. Use translation to $\cdot$ and back.
4. Take

$$Ux = \text{IF } Z_?x \text{ THEN } (Z_{\mathtt{K}}x)\underline{\mathtt{K}}\,\underline{\mathtt{S}} \text{ ELSE } A_s(U(P_0x))(U(P_1x))$$

   where A_s is a combinator witnessing axiom A for the standard coding.
   Then take

$$\mathtt{E} = \mathtt{E}_s \circ U, \qquad \Delta = \Delta_s \circ U$$

   where $\mathtt{E}_s$ and Δ_s are the evaluator and the discriminator for the the standard coding which were constructed in part 1. ◀

3.2 Negative results

Notice that when the coding $\ulcorner\cdot\urcorner$ is a constant map, then it satisfies CON but neither Z nor E. Thus, as pointed out by an anonymous referee, two of Smullyan's questions have trivial answers.

A slight modification to the standard coding gives a counterexample that is also monic and solvable.

▶ **Theorem 9.** *There exists a map* $\ulcorner\cdot\urcorner$ *which satisfies* CON, MON, SOL, *and* P, *yet neither* Z *nor* E. *In particular,* CON $\not\Longrightarrow$ DES.

Proof. Define $\ulcorner\cdot\urcorner$ by

- $\ulcorner M \urcorner \equiv [\Omega M]$ if $M \in \{\mathtt{K}, \mathtt{S}\}$
- $\ulcorner MN \urcorner \equiv \mathtt{P}\ulcorner M \urcorner\ulcorner N \urcorner = [\ulcorner M \urcorner, \ulcorner N \urcorner]$

Note that $\ulcorner\cdot\urcorner$ is monic, solvable, and satisfies A, P, and $\mathrm{Z}_?$ via the same combinators as the standard coding. The combinator witnessing B must be modified ever so slightly:

$$Bx = \text{IF } Z_?x \text{ THEN } (Z_{\mathtt{K}}x)^{\ulcorner\ulcorner\mathtt{K}\urcorner\ulcorner\ulcorner\mathtt{S}\urcorner\urcorner} \text{ ELSE } \mathtt{P}(\mathtt{P}^{\ulcorner\mathtt{P}\urcorner}(B(P_0x)))(B(P_1x))$$

To finish the proof, note that if $Z(\lambda x.x(\Omega M)) = \mathtt{K}$, then by Genericity Lemma [1, 14.3.24] we have $Z(\lambda x.x(\Omega M)) = Z(\lambda x.x(\Omega N))$. Therefore, no term can satisfy axiom Z. By the same token, no evaluator can exist, for its value on $\ulcorner\mathtt{K}\urcorner$ would necessarily agree with that on $\ulcorner\mathtt{S}\urcorner$. ◀

▶ **Theorem 10.** *There exists a map $\ulcorner\cdot\urcorner$ satisfying* Z *and* P *which does not satisfy* A. *Thus* DES $\not\Rightarrow$ CON. *Furthermore, $\ulcorner\cdot\urcorner$ is monic and solvable.*

Proof. For $M \in \mathcal{C}$, let $s(M)$ denote the size of the syntax tree of M, defined inductively by $s(\mathtt{K}) = s(\mathtt{S}) = 1,\ s(MN) = 1 + s(M) + s(N)$. Certainly, $s(M)$ can be easily computed from $\underline{M}$:

$$\tilde{s}x = \text{IF } Z_?x \text{ THEN } \mathtt{c}_1 \text{ ELSE } \mathtt{c}_+\mathtt{c}_1(\mathtt{c}_+(\tilde{s}(P_0x))(\tilde{s}(P_1x)))$$

where $\mathtt{c}_n$ is the n'th Church numeral, and $\mathtt{c}_+$ denotes addition.

For $n \in \mathbb{N}$, let H_n be a lambda term encoding the first n values of the characteristic function of the halting problem. Specifically, we put $H_n = \langle h_0, h_1, \ldots, h_{n-1}\rangle = \lambda z.z\vec{h}$, where

$$h_i = \begin{cases} \mathtt{K} & \varphi_i(i)\downarrow \\ \overline{\mathtt{K}} & \text{otherwise} \end{cases}$$

For $0 \le k \le n$, let Π^n_k be such that $\Pi^n_k\langle M_1, \ldots, M_n\rangle = \langle M_1, \ldots, M_k\rangle$. For example, Π^n_k could be obtained by taking $\Pi^n_k = \Pi\mathtt{c}_n\mathtt{c}_k$, where

$$\Pi nk = \lambda x.\mathtt{B}x(k\mathtt{B}(\lambda s.k\langle\mathtt{I}\rangle(n\mathtt{K}s)))$$

(Here $\mathtt{B}$ is the composition combinator $\lambda xyz.x(yz)$.)

Finally, put

$$\ulcorner M\urcorner = [\underline{M}, H_{s(M)}] \tag{1}$$

Trivially, MON and SOL are satisfied. To see that this coding satisfies axiom Z, we simply compose the combinator Z_b for the standard coding with the first pair-projection:

$$(\lambda x.Z_b(x\mathtt{K}))^{\ulcorner M\urcorner} = \begin{cases} \mathtt{K} & M \equiv b \\ \overline{\mathtt{K}} & \text{otherwise} \end{cases} \qquad b \in \{\mathtt{K}, \mathtt{S}\}$$

For the ith projection, we use the standard combinator P_i with the auxiliary combinators we defined above:

$$\begin{aligned}
&(\lambda x.(\lambda y.[y, \Pi(\tilde{s}(x\mathtt{K}))(\tilde{s}y)(x\overline{\mathtt{K}})])(P_i(x\mathtt{K})))^{\ulcorner M_0M_1\urcorner} \\
=&(\lambda y.[y, \Pi(\tilde{s}(\ulcorner M_0M_1\urcorner\mathtt{K}))(\tilde{s}y)(\ulcorner M_0M_1\urcorner\overline{\mathtt{K}})])(P_i(\ulcorner M_0M_1\urcorner\mathtt{K})) \\
=&(\lambda y.[y, \Pi(\tilde{s}(\ulcorner M_0M_1\urcorner\mathtt{K}))(\tilde{s}y)(\ulcorner M_0M_1\urcorner\overline{\mathtt{K}})])(P_i([\underline{M_0M_1}, H_{s(M_0M_1)}]\mathtt{K})) \\
=&(\lambda y.[y, \Pi(\tilde{s}(\ulcorner M_0M_1\urcorner\mathtt{K}))(\tilde{s}y)(\ulcorner M_0M_1\urcorner\overline{\mathtt{K}})])(P_i\underline{M_0M_1}) \\
=&(\lambda y.[y, \Pi(\tilde{s}\underline{M_0M_1})(\tilde{s}y)(H_{s(M_0M_1)})])\underline{M_i} \\
=&[\underline{M_i}, \Pi(\tilde{s}\underline{M_0M_1})(\tilde{s}\underline{M_i})\langle h_0, \ldots, h_{s(M_0M_1)-1}\rangle] \\
=&[\underline{M_i}, \langle h_0, \ldots, h_{s(M_i)-1}\rangle] \\
=&\ulcorner M_i\urcorner
\end{aligned}$$

Hence (1) satisfies DES. But notice that

$$\varphi_e(e)\downarrow \iff \ulcorner \mathtt{K}^e\mathtt{I}\urcorner\overline{\mathtt{K}}\mathtt{U}^e_e = \mathtt{K}$$

Therefore, if there was a combinator for axiom A, we could decide the halting problem by checking whether $\mathtt{c}_e(A\ulcorner\mathtt{K}\urcorner)\ulcorner\mathtt{I}\urcorner\overline{\mathtt{K}}\mathtt{U}^e_e$ equals $\mathtt{K}$ or $\overline{\mathtt{K}}$. Such an A cannot exist. Thus $\ulcorner\cdot\urcorner$ does not satisfy CON. ◀

Notice that non-computability of the coding $\ulcorner\cdot\urcorner$ was essential in the proof above. Indeed, if the coding was computable, then axiom U^{-1} would be satisfied. By Proposition 8, part 4, the destructor axiom would make the coding complete. Then by part 3, it would satisfy CON.

3.3 Positive results

The next natural question is what additional property could be sufficient for the equivalence CON $\iff$ DES to hold. It turns out that existence of a discriminator goes quite far in this direction.

▶ **Theorem 11.** $\Delta \wedge \mathrm{U}^{-1} \Longrightarrow \mathrm{U}$.

Proof. To construct U, we need to uniformly enumerate all combinators built up from $\mathtt{K}$ and $\mathtt{S}$. Recall that $[M_n]$ is a *uniform enumeration* of $\{M_n\}$ if for each k, there is some X_k such that

$$[M_n] = [M_0, [M_1, [M_2, \dots [M_k, X_k] \dots]$$

That is, $[M_n]$ is an infinite stream whose elements form the sequence $\{M_n\}$. The following functions operate on streams:

$$\begin{aligned}
\mathtt{Map}fm &= [f(m\mathtt{K}), \mathtt{Map}f(m\overline{\mathtt{K}})] \\
\mathtt{Fold}fm &= f(m\mathtt{K})(\mathtt{Fold}f(m\overline{\mathtt{K}})) \\
\mathtt{Merge}mn &= [m\mathtt{K}, [n\mathtt{K}, \mathtt{Merge}(m\overline{\mathtt{K}})(n\overline{\mathtt{K}})]]
\end{aligned}$$

(These definitions implicitly make use of fixed-point combinators.)

Now we define the *standard enumeration* of CL terms to be

$$\mathcal{C} = [\mathtt{K}, [\mathtt{S}, \mathtt{Fold}\ \mathtt{Merge}\ (\mathtt{Map}\ (\lambda s.\mathtt{Map}\ s\ \mathcal{C})\ \mathcal{C})]]$$

It is straightforward to verify that for each $M \in \mathcal{C}$ there is a unique n such that $M \equiv C_n$, where $\mathcal{C} = [C_0, [C_1, \dots]]$. But here we need the combinators to be quoted, hence we define

$$\underline{\mathcal{C}} = [\underline{\mathtt{K}}, [\underline{\mathtt{S}}, \mathtt{Fold}\ \mathtt{Merge}\ (\mathtt{Map}\ (\lambda s.\mathtt{Map}\ (\mathtt{P}s)\ \underline{\mathcal{C}})\ \underline{\mathcal{C}})]]$$

where $\mathtt{P}$ is the pairing combinator from Definition 3. (Note that this notation is overloaded; we don't mean that $\underline{\mathcal{C}}$ is the standard quote of $\mathcal{C}$.)

Define

$$U_0sx = \textsc{if}\ \Delta x(U^{-1}(s\mathtt{K}))\ \textsc{then}\ s\mathtt{K}\ \textsc{else}\ U_0(s\overline{\mathtt{K}})x$$

$$U = U_0\underline{\mathcal{C}}$$

Note that $U_0\underline{\mathcal{C}}\ulcorner M\urcorner = \underline{M}_n$, where $n = (\mu k)(M \equiv M_k \in \mathcal{C})$. Thus

$$U\ulcorner M\urcorner \equiv \underline{M}.$$

This completes the proof of the theorem. ◀

▶ **Theorem 12.** *Suppose* $\ulcorner\cdot\urcorner$ *is a coding which satisfies* Δ*. Then* $\mathrm{U}^{-1} \iff \mathrm{A}$*. In particular,* $\mathrm{CON} \wedge \Delta \implies \mathrm{DES}$*.*

Proof. ($\Rightarrow$) By the theorem above and Proposition 8.3,
$\Delta \wedge \mathrm{U}^{-1} \implies \mathrm{CMP} \implies \mathrm{CON} \wedge \mathrm{DES} \implies \mathrm{A}$.

($\Leftarrow$) Suppose Δ and A are satisfied. Put

$$\begin{aligned} U^{-1}x = \quad & \textsc{if } Z_? x \\ & \textsc{then if } Z_{\mathtt{K}} x \textsc{ then } \ulcorner \mathtt{K} \urcorner \textsc{ else } \ulcorner \mathtt{S} \urcorner \\ & \textsc{else } A(U^{-1}(P_0 x))(U^{-1}(P_1 x)) \end{aligned}$$

By induction, $U^{-1}\underline{M} = \ulcorner M \urcorner$, hence U^{-1} is satisfied. ◀

It remains to consider the question of reconstructing the quote from the Destructor axioms. The problem with using the approach of Theorem 11, where we try to "guess" the quote of a term by comparing every possibility to the input, is that we have no information on the space of these possibilities. This is where the range test comes in. Recall that the statement of this axiom is

$$\exists \text{ c.e. } D \subseteq \mathcal{C},\ \mathsf{Range}(\ulcorner\cdot\urcorner) \subseteq D,\ \exists R_D \in \mathcal{C},\ \forall N \in D : \\ R_D N = \begin{cases} \mathtt{K} & \exists M.\ N = \ulcorner M \urcorner \\ \overline{\mathtt{K}} & \text{otherwise} \end{cases}$$

With the axiom above, we state the final theorem.

▶ **Theorem 13.** $\mathrm{DES} \wedge \mathrm{R}_D \implies \mathrm{U}^{-1}$.

Proof. As in the proof of Theorem 11, we construct $U^{-1}x$ by looking at all possibilities until we find one that matches x, according to the standard discriminator. Let D enumerate a superset of $\mathsf{Range}(\ulcorner\cdot\urcorner)$. We receive this fact as a uniform enumeration $D = [M_n]$, such that for each n one has $R_D M_n = \mathtt{K}$ if $M_n = \ulcorner N \urcorner$ and $R_D M_n = \overline{\mathtt{K}}$ otherwise. Furthermore, every quote $\ulcorner N \urcorner$ appears in the list D: $\forall N \exists n\, \ulcorner N \urcorner = M_n$.

As per Proposition 8.4, let U witness axiom U for $\ulcorner\cdot\urcorner$. Now put

$$U^{-1}x = \mathtt{Fold}\ (\lambda ht.\ \textsc{if } (R_D h)(\Delta x(Uh))\overline{\mathtt{K}} \textsc{ then } h \textsc{ else } t)\ D$$

It is routine to verify that $U^{-1}\underline{M} = \ulcorner M \urcorner$ for each M. ◀

▶ **Corollary 14.** *For a complete coding, one of the following suffices:*

$$\begin{aligned} &\mathrm{A} \wedge \Delta \\ &\mathrm{U}^{-1} \wedge \Delta \\ &\mathrm{U}^{-1} \wedge \mathrm{DES} \\ &\mathrm{R}_D \wedge \mathrm{DES} \end{aligned}$$

Proof. By the theorems 8 through 13. ◀

4 An application: minimal codings

We conclude by applying the above results to a practical problem concerning minimal codings.

Below we define what is probably the simplest non-trivial coding in Combinatory Logic. According to Barendregt, it was first suggested by Dana Scott in a letter to Troelstra. [4]

▶ **Definition 15.** (Scott's coding) Let
- $\ulcorner b \urcorner = \mathtt{K}b$, $b \in \{\mathtt{K}, \mathtt{S}\}$
- $\ulcorner MN \urcorner = \mathtt{S}\ulcorner M \urcorner \ulcorner N \urcorner$

However, it turns out that Scott's coding is not preadmissible, i.e., does not satisfy DES.

▶ **Proposition 16.** *Scott's coding is not complete.*

Proof. Observe that

$$\begin{aligned}
\ulcorner \Omega \urcorner &= \ulcorner \mathtt{SII}(\mathtt{SII}) \urcorner \\
&= \mathtt{S}\ulcorner \mathtt{SII} \urcorner \ulcorner \mathtt{SII} \urcorner \\
&= \lambda z.(\ulcorner \mathtt{SII} \urcorner z)(\ulcorner \mathtt{SII} \urcorner z) \\
&= \lambda z.(\mathtt{S}\ulcorner \mathtt{SI} \urcorner \ulcorner \mathtt{I} \urcorner z)(\mathtt{S}\ulcorner \mathtt{SI} \urcorner \ulcorner \mathtt{I} \urcorner z) \\
&= \lambda z.(\ulcorner \mathtt{SI} \urcorner z(\ulcorner \mathtt{I} \urcorner z))(\ulcorner \mathtt{SI} \urcorner z(\ulcorner \mathtt{I} \urcorner z)) \\
&= \lambda z.(\mathtt{S}\ulcorner \mathtt{S} \urcorner \ulcorner \mathtt{I} \urcorner z(\mathtt{K}\mathtt{I}z))(\mathtt{S}\ulcorner \mathtt{S} \urcorner \ulcorner \mathtt{I} \urcorner z(\mathtt{K}\mathtt{I}z)) \\
&= \lambda z.(\ulcorner \mathtt{S} \urcorner z(\ulcorner \mathtt{I} \urcorner z)\mathtt{I})(\ulcorner \mathtt{S} \urcorner z(\ulcorner \mathtt{I} \urcorner z)\mathtt{I}) \\
&= \lambda z.(\mathtt{K}\mathtt{S}z(\mathtt{K}\mathtt{I}z)\mathtt{I})(\mathtt{K}\mathtt{S}z(\mathtt{K}\mathtt{I}z)\mathtt{I}) \\
&= \lambda z.\mathtt{SII}(\mathtt{SII})
\end{aligned}$$

is unsolvable. By Proposition 8.2, $\ulcorner \cdot \urcorner$ does not satisfy Z. Then it also fails to satisfy DES, and is therefore not complete. ◀

Scott's coding is very elegant and minimalistic: the size of $\ulcorner M \urcorner$ is exactly double the size of M, and the number of strong (combinatory) reductions required to bring $\ulcorner M \urcorner \mathtt{I}$ to M is exactly the size of M. It is a shame that this coding is not complete. Could there be a substitute?

▶ **Definition 17.** Let the *minimal coding* be defined by
- $\ulcorner b \urcorner = \mathtt{SI}(\mathtt{K}b) = \underline{b}$, $b \in \{\mathtt{K}, \mathtt{S}\}$.
- $\ulcorner MN \urcorner = \mathtt{S}\ulcorner M \urcorner \ulcorner N \urcorner$

Note that $\ulcorner M \urcorner = \lambda z.M_z$, where M_z is obtained from M by replacing every occurrence of a basic combinator b with zb.

▶ **Proposition 18.** *The minimal coding is complete.*

Proof. The recursive definition makes it straightforward to construct $\ulcorner M \urcorner$ once the standard code of M is given:

$$U^{-1}(\underline{M}) = \text{IF } Z_? \underline{M} \text{ THEN } \underline{M} \text{ ELSE } \mathtt{S}(U^{-1}(P_0\underline{M}))(U^{-1}(P_1\underline{M}))$$

(with P_i referring to the standard projections.)

For the opposite direction, we use the characterization result to infer the existence of a combinator satisfying U. By Corollary 14, having already defined U^{-1}, all that remains is to satisfy DES.

Let W and Z be the following terms:

$W = \lambda wnm.[w, [m\mathtt{S}, n\mathtt{S}]]$

$Z = \lambda x.[W, \mathtt{K}\langle x\rangle]$

By induction we will show that

$$\ulcorner M\urcorner Z = [W, M'], \qquad \text{with } M'\mathtt{S} = \ulcorner M\urcorner \tag{2}$$

Base case: If $b \in \{\mathtt{K}, \mathtt{S}\}$, then $\ulcorner b\urcorner Z = Zb = [W, \mathtt{K}\langle b\rangle]$. Furthermore,

$$\mathtt{K}\langle b\rangle\mathtt{S} = \langle b\rangle = \ulcorner b\urcorner$$

Induction: We compute

$$\begin{aligned}
\ulcorner MN\urcorner Z &= \mathtt{S}\ulcorner M\urcorner\ulcorner N\urcorner Z \\
&= \ulcorner M\urcorner Z(\ulcorner N\urcorner Z) \\
&=_{\text{IH}} [W, M'][W, N'] \\
&= [W, N']WM' \\
&= WWN'M' \\
&= [W, [M'\mathtt{S}, N'\mathtt{S}]] \\
&=_{\text{IH}} [W, [\ulcorner M\urcorner, \ulcorner N\urcorner]]
\end{aligned}$$

Furthermore,

$$[\ulcorner M\urcorner, \ulcorner N\urcorner]\mathtt{S} = \mathtt{S}\ulcorner M\urcorner\ulcorner N\urcorner = \ulcorner MN\urcorner$$

Note that, with (2) verified, the proof of the induction step also gives us that $\ulcorner MN\urcorner Z = [W, [\ulcorner M\urcorner, \ulcorner N\urcorner]]$. Hence we can satisfy P by taking the terms $P_i = \lambda c.cZ\mathtt{U}_1^1\mathtt{U}_i^1$.

We can also separate application nodes from the leaves by the term

$$Z_?c = cZ\mathtt{U}_1^1(\lambda mno.\overline{\mathtt{K}})(\mathtt{KK})$$

Finally, $Z_?$ can be used to satisfy Z exactly as in the case of standard coding, by appealing to Böhm's theorem. (Using the fact that $\ulcorner\mathtt{K}\urcorner$ and $\ulcorner\mathtt{S}\urcorner$ are distinct, closed $\beta\eta$-normal forms.) ◀

▶ **Remark 19.** The reader might wonder whether the minimal coding could be trimmed even further by putting $\ulcorner b\urcorner = \mathtt{SI}b$ for the basic combinators. It turns out that this coding is complete as well, and for the uncoding map one can take $Ux = xZ\mathtt{U}_1^1$, where $Zx = [W, x(\mathtt{K}^4\underline{\mathtt{I}})(\mathtt{K}^3\underline{\mathtt{S}})(\mathtt{K}^2\underline{\mathtt{K}})]$ and $Wwnm = [w, [m, n]]$. However, since the evaluator is significantly more complex, we propose to regard the previous definition as *the* minimal quote for combinators.

References

1 Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 2nd edition, 1984.

2 Henk Barendregt. Discriminating Coded Lambda Terms. In A. Anderson and M. Zeleny, editors, *Logic, Meaning, and Computation: Essays in Memory of Alonzo Church*, volume 305 of *Synthese Library*. Springer, 1994.

3 Henk Barendregt. Enumerators of Lambda Terms Are Reducing Constructively. In *Dirk van Dalen Festschrift*, volume 5 of *Questionnes Infinitae*. Utrecht University, Department of Philosophy, 1999.

4 Henk Barendregt, June 2011. Private communication.

5 Alessandro Berarducci and Corrado Böhm. A self-interpreter of lambda calculus having a normal form. In Egon Börger, Gerhard Jäger, Hans Kleine Büning, Simone Martini, and Michael M. Richter, editors, *CSL*, volume 702 of *Lecture Notes in Computer Science*, pages 85–99. Springer, 1992.

6 Douglas R. Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books, 20 anv edition, February 1999.

7 Stephen C. Kleene. Lambda-definability and recursiveness. *Duke Mathematical Journal*, 2:340–353, 1936.

8 John McCarthy. *LISP 1.5 Programmer's Manual*. The MIT Press, 1962.

9 Torben Mogensen. Efficient Self-Interpretation in Lambda Calculus. *Journal of Functional Programming*, 2:345–364, 1994.

10 Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *PLDI*, pages 199–208, 1988.

11 Brian Cantwell Smith. *Procedural Reflection in Programming Languages, Volume I*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1982.

12 Raymond Smullyan. *Diagonalization and Self-Reference*. Oxford University Press, USA, 1994.

Relative Completeness for Logics of Functional Programs

Bernhard Reus [1] **and Thomas Streicher** [2]

1 **University of Sussex,**
Brighton BN1 9QH, UK
bernhard@sussex.ac.uk
2 **TU Darmstadt,**
64289 Darmstadt, Germany
streicher@mathematik.tu-darmstadt.de

Abstract

We prove a relative completeness result for a logic of functional programs extending D. Scott's LCF. For such a logic, contrary to results for Hoare logic, it does not make sense to ask whether it is complete relative to the full theory of its first-order part, since the first order part does not determine uniquely the model at higher-order types. Therefore, one has to fix a model and choose an appropriate data theory w.r.t. which the logic is relatively complete. We establish relative completeness for two models: for the Scott model we use the theory of Baire Space as data theory, and for the effective Scott model we take first-order arithmetic. In both cases we need to extend traditional LCF in order to capture a sufficient amount of domain theory.

1998 ACM Subject Classification D.2.4 Program Verification; D.3.1 Formal Definitions and Theory; F.3.1 Specifying and Verifying and Reasoning about Programs; F.3.2 Semantics of Programming Languages

Keywords and phrases Completeness, Program Logics, LCF

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.470

1 Introduction

Program logics play an important role in Computer Science to complement testing. A program logic allows one to prove that a program satisfies a given specification. Seminal work has been done in the late sixties by Hoare on axiomatic semantics for stateful programs [8]. Since then many calculi have been developed for all kinds of programming languages and meanwhile mechanizations of these logics in numerous verification tools exist.

Two properties of a program logic are of particular interest. *Soundness* states that any property one can prove of a program in the calculus is actually valid. *Completeness* states the converse, namely that any valid property can also be derived. In an ideal world, a formal calculus for a program logic would be both, sound *and* complete, thus faithfully and completely reflecting the semantics of programs and correctness assertions, also called specifications.

Due to Gödel's Incompleteness Theorem it is hopeless to look for absolutely complete program logics since for any (sufficiently expressive) formal system S one can find a correctness assertion G_S which is true but cannot be derived in S. Nevertheless one might ask whether the axioms of some program logic $\mathcal{L}$ are sufficient for proving all true correctness assertions relative to some complete data theory $\mathcal{T}$, i.e. whether $\mathcal{L}$ is *complete relative* to $\mathcal{T}$.

In [6] this problem was considered for the case where $\mathcal{L}$ is the Hoare logic for a basic imperative language which can store and manipulate objects of a data structure and $\mathcal{T}$ is the

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 470–480

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

complete first order theory of this data structure. Obviously, the logic $\mathcal{L}$ is complete relative to $\mathcal{T}$ provided that for every program P and postcondition B

(a) the *weakest liberal precondition* $wlp(P)(B)$ is expressible in $\mathcal{T}$ and
(b) $\{wlp(P)(B)\}P\{B\}$ is provable in $\mathcal{L}$

because by definition $\{A\}P\{B\}$ is equivalent to $A \Rightarrow wlp(P)(B)$ and $\mathcal{L}$ can derive $\{A\}P\{B\}$ from $A \Rightarrow wlp(P)(B)$ and $\{wlp(P)(B)\}P\{B\}$ via the consequence rule. In [6] it was shown that (b) holds under the assumption of (a), i.e. that $\mathcal{T}$ is *expressive* w.r.t $\mathcal{L}$. In practice, expressivity is ensured by the first order definability of $[\![P]\!]$, the semantics of P: if R is a first order relation expressing $[\![P]\!]$ then $wlp(P)(B)(s)$ can be expressed as $\forall s'.\ R(s,s') \Rightarrow B(s')$. A typical example is obtained by taking for $\mathcal{T}$ the set of all *true* first order sentences of arithmetic since for all programs P its input/output relation $[\![P]\!]$ is recursively enumerable and thus expressible by a formula of first order arithmetic.

To the best of our knowledge, the question of relative completeness for logics of *functional* programming languages has not been investigated so far[1] though it has been suggested in [13].

Historically, the first logic for a functional programming language was Dana Scott's LCF introduced in [26]. This is a (many-sorted) predicate logic whose terms are PCF programs as studied in [20] and many subsequent publications. There is some pragmatic evidence that most correctness assertions about PCF programs can be proved within LCF. But there are quite easy correctness assertions which can neither be proved nor disproved in LCF. Let, for example, $E(f)$ be the purely equational specification of the "parallel or" function then LCF proves neither $\exists f.E(f)$ nor its negation. The reason simply is that the former holds in the Scott model but its negation holds in the fully abstract model (cf. [17]) where "parallel or" does not exist (see [20]). Notice, however, that these two models are not different w.r.t. the data type `nat` of natural numbers (and the type `nat`→`nat` of unary functions on `nat`) but they do differ at higher types and, actually, already at type `nat`→`nat`→`nat`, the type of "parallel or". Accordingly, it does not make sense to ask whether LCF is relatively complete w.r.t. the *full theory of its first order part* since the latter – unlike for the basic imperative language considered in [6] – does not fully determine the (higher type part of the) model.

Thus, the right question is whether "natural" models for PCF can have nice axiomatizations $\mathcal{L}$ which are complete relative to a full data theory $\mathcal{T}$. Though "natural" is somewhat subjective one may want to consider the following three kinds of models:

(1) the Scott model and its effective variant (for an introduction see e.g. [27])
(2) the observably sequential algorithms model and its effective variant (see the original paper [4] or more modern adaptations like [10, 14, 15])
(3) fully abstract models like Milner's model (see [17, 27]) or its sequentially realised submodel $\mathcal{F}$ (see [18]).

The models in (1) allow one to interpret PCF^{++}, i.e. PCF extended with a parallel or and a continuous existential quantifier as in [20]. The models in (2) allow one to interpret SPCF, an extension of PCF with error elements and a `catch`-construct which allows one to observe sequentiality (see [4, 15]). In both cases all types σ appear as definable retracts of type `nat`→`nat`. Thus, it seems plausible that one can axiomatize these models by adding some "obvious" axioms to LCF (of course, depending on the kind of model) and show completeness

[1] A notable exception is [9] where a different form of completeness for a functional language with state is proven that is weaker (see Conclusions).

of the ensuing logics, relative to the complete theory $\mathcal{T}$ of the total strict elements of $\mathtt{nat}\to\mathtt{nat}$. In case of effective variants of these models it is, however, possible to instantiate $\mathcal{T}$ by the set of all arithmetic truths because it suffices to consider the computable elements of every type which can be encoded by natural numbers (see e.g. [20]).

In this paper we will perform the task for (1) in Sections 2 and 3, respectively. In the final section we will discuss the cases (2) and (3) and extensions to models with higher order references.

2 The Scott Model

Let $\mathcal{D}$ be the Scott model of PCF as introduced in [26]. In *loc.cit.* one finds a program logic LCF suitable for reasoning about elementary properties of PCF programs (see also Sect. 3.3 of [27] for a quick recap of LCF). However, the axioms of LCF are so general that they hold in all cpo-enriched order extensional models of PCF. The aim of this paper is to extend LCF to a logic $\mathcal{L}$ for which $\mathcal{D}$ is a model and which is complete relative to the complete theory $\mathcal{T}$ of Baire space $\mathbb{N}^{\mathbb{N}}$ (considered as a subset of the interpretation of $\mathtt{nat}\to\mathtt{nat}$ in the Scott model). This theory $\mathcal{T}$ will be modeled after the theory **EL** (short for *Elementary Analysis*) of [28] which is "an extension of Heyting arithmetic with variables and quantifiers for number-theoretic functions". Our theory $\mathcal{T}$ differs from **EL** in two respects: it is based on classical logic and formulated in a sublanguage of $\mathcal{L}$ which refers only to the strict total elements of $\mathtt{nat}$ and $\mathtt{nat}\to\mathtt{nat}$. In order to stay within the realm of $\mathbb{N}^{\mathbb{N}}$, general recursion is not available in the language of $\mathcal{T}$ though primitive recursion is. But this is not a real restriction since all inhabited r.e. sets can be enumerated by a primitive recursive function. This fact will be used subsequently without further mention.

As shown in Plotkin's paper [21] every coherently complete countably algebraic domain appears as retract of $[\mathbb{N}_\perp\to\mathbb{N}_\perp]$, the interpretation of $\mathtt{nat}\to\mathtt{nat}$ in $\mathcal{D}$. By inspection of the proof in [21] one sees that $\mathtt{nat}\to\mathtt{nat}$ contains all PCF types as computable retracts of $\mathtt{nat}\to\mathtt{nat}$. Thus, from results in [20] it follows that for every PCF type σ there exist PCF^{++} programs $\mathtt{e}_\sigma : \sigma \to \mathtt{nat} \to \mathtt{nat}$ and $\mathtt{p}_\sigma : (\mathtt{nat} \to \mathtt{nat}) \to \sigma$ such that $\mathtt{p}_\sigma \circ \mathtt{e}_\sigma$ is the identity on D_σ (the interpretation of type σ in $\mathcal{D}$). By PCF^{++} we denote the extension of PCF with the "parallel or" operation $\mathsf{por} : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}$ and Plotkin's continuous existential quantifier $\exists : (\mathtt{nat} \to \mathtt{nat}) \to \mathtt{nat}$. As shown in [20] (see also final chapter of [27]) all computable elements of the Scott model arise as denotations of PCF^{++}-terms. Recall that an element $d \in D_\sigma$ is computable if, and only if, the set of codes of compact approximations to d is recursively enumerable by some (prim. rec.) function α_d. In the language $\mathcal{T}$ we can refer to elements of D_σ in terms of sequences $\alpha \in \mathbb{N}^{\mathbb{N}}$ which enumerate codes of compact approximations to d. See [3, 2] for further information on representations of domains and more general spaces via Baire space and its connection to function realizability.

We will define a program logic $\mathcal{L}$ which axiomatizes the Scott model sufficiently well. Our aim is to show that $\mathcal{L}$ is complete relative to $\mathcal{T}$. For this purpose it suffices that for every $\mathcal{L}$-predicate P on objects of type σ

(A) there is a $\mathcal{T}$-predicate $\widetilde{P}$ such that $P(M)$ is equivalent to $\widetilde{P}(\alpha_M)$ for all closed PCF terms M of type σ and
(B) $\mathcal{L}$ proves that $P(M) \leftrightarrow \widetilde{P}(\alpha_M)$

where α_M enumerates the codes of compact approximations to (the interpretation of) M. Condition (A) is analogous to the expressivity condition (a) in Cook's original proof since (A) requires that every specification P formulated in the program logic $\mathcal{L}$ can be expressed

equivalently by a predicate $\widetilde{P}$ in the "data theory" $\mathcal{T}$. Condition (B) is analogous to condition (b) in Cook's original proof since (B) requires that the program logic is strong enough to prove this equivalence.

2.1 The program logic $\mathcal{L}$

The logic $\mathcal{L}$ is similar to the language of LCF as introduced in [26] in the respect that its base types are the types of PCF. However, terms of type σ will be all PCF^{++}-terms. The only base predicates are the inequality relations $\sqsubseteq_\sigma$ on type σ. Equality on σ can be expressed as $x \sqsubseteq_\sigma y \wedge y \sqsubseteq_\sigma x$. In contrast to the original LCF, our $\mathcal{L}$ will not be based on classical first order but rather on *classical higher order predicate logic*.

The usual axioms of LCF are sufficient for performing most simple verification exercises but do not capture the deeper domain theoretic structure of the Scott model. From our axiomatization one can derive all the axioms of traditional LCF but we also will axiomatize a reasonable part of domain theory à la Scott. This was done also in [22] and was machine checked within HOL (a "synthetic" intuitionistic version has been developed and verified in [23] using LEGO). Unlike those formalizations, however, we will also have to speak about compact elements. In order to do that we do not need to extend the term language but we need to state continuity and similar properties in terms of compact elements.

First of all we postulate that all relations $\sqsubseteq_\sigma$ are partial orders. Furthermore, using higher order logic we can state that all types σ are complete partial orders w.r.t. $\sqsubseteq_\sigma$.

(1) every type σ is a directed complete partial order (dcpo)

Furthermore, we require that

(2) all f of type $\sigma{\to}\tau$ are Scott continuous

The next two axioms characterize the order and suprema in function spaces

(3) for all f, g of type $\sigma{\to}\tau$ we have $f \sqsubseteq_{\sigma\to\tau} g$ iff $\forall x{:}\sigma.\, f(x) \sqsubseteq_\tau g(x)$
(4) for all directed subsets F of $\sigma{\to}\tau$ we have $\forall x{:}\sigma.\, \big(\bigsqcup F\big)(x) = \bigsqcup_{f\in F} f(x)$

where $\bigsqcup F$ is the supremum of F in $\sigma{\to}\tau$ whose existence follows from axiom (1). The following axioms (5–9) are standard:

(5) $\lambda x{:}\sigma.M_1 \sqsubseteq_{\sigma\to\tau} \lambda x{:}\sigma.M_2 \Leftrightarrow \forall x{:}\sigma.\, M_1 \sqsubseteq_\tau M_2$
(6) $(\lambda x{:}\sigma.M)(N) =_\tau M[N/x]$
(7) $\lambda x{:}\sigma.M(x) =_{\sigma\to\tau} M$ provided x is not free in M
(8) $\mathsf{fix}_\sigma(M) =_\sigma M(\mathsf{fix}_\sigma(M))$
(9) $\forall x{:}\sigma.\, M(x) \sqsubseteq_\sigma x \Rightarrow \mathsf{fix}_\sigma(M) \sqsubseteq_\sigma x$ provided x is not free in M

Thus, for $\Omega_\sigma \equiv \mathsf{fix}_\sigma(\lambda x{:}\sigma.x)$ we can show that $\forall x{:}\sigma.\Omega_\sigma \sqsubseteq_\sigma x$. Further, we postulate axiom

(10) forall f of type $\sigma{\to}\sigma$ we have $\mathsf{fix}_\sigma(f) = \bigsqcup_{n\in\omega} f^n(\Omega_\sigma)$.

from which one can derive fixpoint induction and computational induction as usual.

Using the defined predicate $N(x) \equiv x \neq \Omega_{\mathtt{nat}}$ we can state the following axioms about `nat`.

(11) $\neg\,\mathsf{succ}(x) = 0$
(12) $\forall x, y{:}\mathtt{nat}.\, N(x) \wedge N(y) \wedge \mathsf{succ}(x) = \mathsf{succ}(y) \Rightarrow x = y$

(13) $P(0) \wedge \big(\forall x{:}\mathtt{nat}.\, N(x) \wedge P(x) \Rightarrow P(\mathsf{succ}(x))\big) \Rightarrow \forall x{:}\mathtt{nat}.\, N(x) \Rightarrow P(x)$
(14) $N(0)$
(15) $\forall x{:}\mathtt{nat}.\, N(x) \Leftrightarrow N(\mathsf{succ}(x))$
(16) $\mathsf{pred}(0) = 0$
(17) $\forall x{:}\mathtt{nat}.\, \mathsf{pred}(\mathsf{succ}(x)) = x$
(18) $\mathsf{ifz}(\Omega_{\mathtt{nat}}, x, y) = \Omega_{\mathtt{nat}}$
(19) $\mathsf{ifz}(0, x, y) = x$
(20) $\forall z{:}\mathtt{nat}.\, N(z) \Rightarrow \mathsf{ifz}(\mathsf{succ}(z), x, y) = y.$

We have to add axioms for the extra PCF^{++} constants por and $\exists$.

(21) $\forall x, y{:}\mathtt{nat}.\, \mathsf{por}(x, y) = \Omega_{\mathtt{nat}} \vee \mathsf{por}(x, y) = 0 \vee \mathsf{por}(x, y) = 1$
(22) $\forall x, y{:}\mathtt{nat}.\, \mathsf{por}(x, y) = 0 \leftrightarrow (x = 0 \vee y = 0)$
(23) $\forall x, y{:}\mathtt{nat}.\, \mathsf{por}(x, y) = 1 \leftrightarrow (x = 1 \wedge y = 1)$
(24) $\forall f{:}\mathtt{nat}{\to}\mathtt{nat}.\, \exists(f) = \Omega_{\mathtt{nat}} \vee \exists(f) = 0 \vee \exists(f) = 1$
(25) $\forall f{:}\mathtt{nat}{\to}\mathtt{nat}.\, \exists(f) = 0 \leftrightarrow \exists x : \mathtt{nat}. N(x) \wedge f(x) = 0$
(26) $\forall f{:}\mathtt{nat}{\to}\mathtt{nat}.\, \exists(f) = 1 \leftrightarrow f(\Omega) = 1$

Though $\mathcal{L}$ is sufficient for expressing "ordinary" correctness proofs it is not clear how to formalize basic domain theory in this language. For this purpose one has to speak about compact elements. For every type σ one can define in PCF^{++} a strict function $\varepsilon^\sigma : \mathtt{nat} \to \sigma$ enumerating the compact elements of D_σ in such a way that $(D_\sigma, \varepsilon^\sigma)$ is an effectively given domain, see [20, 27]. We often write $x \in \varepsilon^\sigma_n$ instead of $\varepsilon^\sigma_n \sqsubseteq x$. The ε^σ are chosen in such a way that

(27) $x \in \varepsilon^{\mathtt{nat}}_0$ and $\forall x{:}\mathtt{nat}.\, x \in \varepsilon^{\mathtt{nat}}_{n+1} \Leftrightarrow x = n$
(28) $f \in \varepsilon^{\sigma\to\tau}_{[\langle n_1, m_1\rangle, \ldots, \langle n_k, m_k\rangle]} \Leftrightarrow \bigwedge_{i=1}^{k} f(\varepsilon^\sigma_{n_i}) \in \varepsilon^\tau_{m_i}$

where $\langle -, -\rangle$ refers to some primitive recursive coding of pairs and $[n_1, \ldots, n_k]$ is a code for the finite set $\{n_1, \ldots, n_k\}$. In order to relate ε^σ to $\sqsubseteq_\sigma$ we postulate the axiom

(29) $x \sqsubseteq_\sigma y \Leftrightarrow \forall n{:}\mathbb{N}.\, x \in \varepsilon^\sigma_n \Rightarrow y \in \varepsilon^\sigma_n$

Continuity of all functions in $\sigma{\to}\tau$ is expressed by the axiom

(30) $\forall f{:}\sigma{\to}\tau. \forall x{:}\sigma. \forall n{:}\mathbb{N}.\, f(x) \in \varepsilon^\tau_n \Rightarrow \exists m{:}\mathbb{N}.\, x \in \varepsilon^\sigma_m \wedge \forall x{:}\sigma.\, x \in \varepsilon^\sigma_m \Rightarrow f(x) \in \varepsilon^\tau_n$

In presence of higher order logic it is clear that the above axioms are sufficient for deriving the usual theorems of domain theory à la Scott (see e.g. other axiomatizations of domain theory like Holcf [22]). However, these axioms are not irredundant[2] but sufficient for their purpose.

2.2 $\mathcal{T}$ as a sublanguage of $\mathcal{L}$

There is an obvious translation from the language of **EL** into the language of $\mathcal{L}$ whose image we denote by $\mathcal{T}$. The type of natural numbers in **EL** will be interpreted in $\mathcal{L}$ as the subset of $\mathtt{nat}$ as given by the predicate N. The type of sequences in **EL** will be interpreted in $\mathcal{L}$ as the subset B of strict and total elements of $\mathtt{nat}{\to}\mathtt{nat}$. We write $\forall n{:}\mathbb{N}.\cdots$ as an abbreviation for $\forall n{:}\mathtt{nat}.N(n) \Rightarrow \cdots$ and $\forall \alpha{:}\mathbb{N}^{\mathbb{N}}.\cdots$ as an abbreviation for $\forall \alpha{:}\mathtt{nat}{\to}\mathtt{nat}.B(\alpha) \Rightarrow \cdots$.

The basic operations of **EL** are interpreted by their corresponding basic operations in $\mathcal{L}$. The primitive recursor of **EL** is implemented in terms of the fixpoint operator of $\mathcal{L}$.

[2] For instance, axioms (8) and (9) follow from(10).

2.3 Reducing $\mathcal{L}$ to $\mathcal{T}$

From [21] it follows that there are PCF^{++} terms

$$\mathsf{p}_{\to} : (\mathtt{nat}\to\mathtt{nat}) \to (\mathtt{nat}\to\mathtt{nat}) \to \mathtt{nat}\to\mathtt{nat} \qquad \text{and}$$
$$\mathsf{e}_{\to} : ((\mathtt{nat}\to\mathtt{nat}) \to \mathtt{nat}\to\mathtt{nat}) \to \mathtt{nat} \to \mathtt{nat}$$

such that $\mathsf{p}_{\to} \circ \mathsf{e}_{\to}$ is the identity on $\mathtt{nat}\to\mathtt{nat}$ and this is provable in $\mathcal{L}$. We may exhibit $\mathtt{nat}$ as a retract of $\mathtt{nat}\to\mathtt{nat}$ by putting $\mathsf{p}_{\mathtt{nat}}(f) = f(0)$ and $\mathsf{e}_{\mathtt{nat}}(x)(y) = x$. For function types $\sigma\to\tau$ we define

$$\mathsf{e}_{\sigma\to\tau}(g) = \mathsf{e}_{\to}(\mathsf{e}_\tau \circ g \circ \mathsf{p}_\sigma) \qquad \mathsf{p}_{\sigma\to\tau}(f) = \mathsf{p}_\tau \circ \mathsf{p}_{\to}(f) \circ \mathsf{e}_\sigma$$

exhibiting $\sigma\to\tau$ as a retract of $\mathtt{nat}\to\mathtt{nat}$ provably in $\mathcal{L}$.[3]

However, in general for a computable f of type $\mathtt{nat}\to\mathtt{nat}$ there will not exist a total recursive α with $\mathsf{p}_\sigma(f) = \mathsf{p}_\sigma(\alpha)$. Thus, it seems appropriate to consider in addition a PCF definable map $\mathsf{r} : (\mathtt{nat}\to\mathtt{nat}) \to (\mathtt{nat}\to\mathtt{nat})$ which turns a sequence of codes of compact elements of $\mathtt{nat}\to\mathtt{nat}$ into the supremum of the coded elements in $\mathtt{nat}\to\mathtt{nat}$, i.e. $\mathsf{r}(\alpha) = \bigsqcup_n \varepsilon^{\mathtt{nat}\to\mathtt{nat}}_{\alpha(n)}$, provided this supremum exists. Obviously, the restriction of r to $\mathbb{N}^\mathbb{N}$ is still surjective on $\mathtt{nat}\to\mathtt{nat}$ and, moreover, the corresponding statement $\forall f{:}\mathtt{nat}\to\mathtt{nat}.\exists\alpha{:}\mathbb{N}^\mathbb{N}.\, f = \mathsf{r}(\alpha)$ is provable in $\mathcal{L}$. Thus, for every type σ one can prove in $\mathcal{L}$ that $\forall x{:}\sigma.\exists\alpha{:}\mathbb{N}^\mathbb{N}.\, x = \widetilde{\mathsf{p}}_\sigma(\alpha)$ where $\widetilde{\mathsf{p}}_\sigma = \mathsf{p}_\sigma \circ \mathsf{r}$. Moreover, one can show that $\widetilde{\mathsf{p}}_\sigma$ restricted to $\mathbb{N}^\mathbb{N}$ is an *admissible* representation of D_σ in the sense of [2]. This means that for every (computable) continuous map f from a subset R of $\mathbb{N}^\mathbb{N}$ to D_σ there exists a (computable) continuous map $\phi : R \to \mathbb{N}^\mathbb{N}$ realizing f, i.e. $f = \widetilde{\mathsf{p}}_\sigma \circ \phi$.

Based on these facts one may replace quantifications of the form $Qx{:}\sigma.A(x)$ (where Q stands for $\forall$ or $\exists$) by $Q\alpha{:}\mathbb{N}^\mathbb{N}.A(\widetilde{\mathsf{p}}_\sigma(\alpha))$. Formulas of the latter form are not yet in the fragment $\mathcal{T}$ since $\widetilde{\mathsf{p}}_\sigma$ is not a term of $\mathcal{T}$. Thus, we have to replace atomic formulas $\widetilde{\mathsf{p}}_\sigma(\alpha) \sqsubseteq_\sigma \widetilde{\mathsf{p}}_\sigma(\beta)$ by $\mathcal{L}$-provably equivalent formulas in the language of $\mathcal{T}$. For this purpose we first replace $\widetilde{\mathsf{p}}_\sigma(\alpha) \sqsubseteq_\sigma \widetilde{\mathsf{p}}_\sigma(\beta)$ by $\forall n{:}\mathbb{N}.\widetilde{\mathsf{p}}_\sigma(\alpha) \in \varepsilon^\sigma_n \Rightarrow \widetilde{\mathsf{p}}_\sigma(\beta) \in \varepsilon^\sigma_n$. Thus, it suffices to replace formulas of the form $\widetilde{\mathsf{p}}_\sigma(\alpha) \in \varepsilon^\sigma_n$ by $\mathcal{L}$-provably equivalent formulas $R_\sigma(\alpha, n)$ in the language of $\mathcal{T}$. This is achieved by the following lemma.

▶ **Lemma 1.** *For every* PCF *type* σ *there is a* $\mathcal{T}$*-predicate* $R_\sigma(\alpha, n)$ *such that*

(†) $\qquad R_\sigma(\alpha, n) \Leftrightarrow \widetilde{\mathsf{p}}_\sigma(\alpha) \in \varepsilon^\sigma_n$

is provable in $\mathcal{L}$*.*

Proof. For base type $\mathtt{nat}$, we get by definition of $\varepsilon^{\mathtt{nat}}$ and r that

$$\begin{aligned}
\widetilde{\mathsf{p}}_{\mathtt{nat}}(\alpha) \in \varepsilon^{\mathtt{nat}}_n \quad &\Longleftrightarrow \quad n = 0 \vee \mathsf{p}_{\mathtt{nat}}(\textstyle\bigsqcup_k \varepsilon^{\mathtt{nat}\to\mathtt{nat}}_{\alpha(k)}) = n{-}1 \\
&\Longleftrightarrow \quad n = 0 \vee (\textstyle\bigsqcup_k \varepsilon^{\mathtt{nat}\to\mathtt{nat}}_{\alpha(k)})(0) = n{-}1 \\
&\Longleftrightarrow \quad n = 0 \vee \exists k{:}\mathtt{nat}.\, (\varepsilon^{\mathtt{nat}\to\mathtt{nat}}_{\alpha(k)})(0) = n{-}1 \\
&\Longleftrightarrow \quad n = 0 \vee \exists k{:}\mathtt{nat}.\, \langle 1, n\rangle \in \alpha(k)
\end{aligned}$$

which is a $\mathcal{T}$-predicate. Therefore, we can set $R_{\mathtt{nat}}(\alpha, n) \equiv n = 0 \vee \exists k{:}\mathtt{nat}.\, \langle 1, n\rangle \in \alpha(k)$. Suppose as induction hypothesis that we have achieved our goal for σ and τ already. Now

[3] In [21] Plotkin shows that all coherently complete countably algebraic cpo's arise as retracts of $\mathtt{nat}\to\mathtt{nat}$. But all PCF types get interpreted as coherently complete countably algebraic cpo's in the Scott model.

we can prove in $\mathcal{L}$ that

$$\begin{aligned}
\widetilde{\mathsf{p}}_{\sigma\to\tau}(\alpha) \in \varepsilon^{\sigma\to\tau}_{[\langle n_1,m_1\rangle,\ldots,\langle n_k,m_k\rangle]} &\iff \bigwedge_{i=1}^{k} \widetilde{\mathsf{p}}_{\sigma\to\tau}(\alpha)(\varepsilon^{\sigma}_{n_i}) \in \varepsilon^{\tau}_{m_i} \\
&\iff \bigwedge_{i=1}^{k} \mathsf{p}_\tau(\mathsf{p}_\to(\mathsf{r}(\alpha))(\mathsf{e}_\sigma(\varepsilon^{\sigma}_{n_i}))) \in \varepsilon^{\tau}_{m_i} \\
&\iff \bigwedge_{i=1}^{k} \widetilde{\mathsf{p}}_\tau(\Phi_{\sigma,\tau}(\alpha,n_i)) \in \varepsilon^{\tau}_{m_i} \\
&\iff \bigwedge_{i=1}^{k} R_\tau(\Phi_{\sigma,\tau}(\alpha,n_i),m_i)
\end{aligned}$$

where $\Phi_{\sigma,\tau}(\alpha,n)$ is a term in $\mathcal{T}$ of type $\mathbb{N}^\mathbb{N}$ such that $\mathcal{L}$ proves $\mathsf{p}_\tau(\mathsf{p}_\to(\mathsf{r}(\alpha))(\mathsf{e}_\sigma(\varepsilon^{\sigma}_{n}))) = \widetilde{\mathsf{p}}_\tau(\Phi_{\sigma,\tau}(\alpha,n))$. The term $\Phi_{\sigma,\tau}$ exists because $\widetilde{\mathsf{p}}_\tau$ is an admissible representation of D_τ and every code of an r.e. set can effectively be transformed into a code of a primitive recursive function enumerating it.

According to the equivalence established above, we may now define $R_{\sigma\to\tau}$ inductively as

$$R_{\sigma\to\tau}(\alpha,[\langle n_1,m_1\rangle,\ldots,\langle n_k,m_k\rangle]) \equiv \bigwedge_{i=1}^{k} R_\tau(\Phi_{\sigma,\tau}(\alpha,n_i),m_i)$$

which can be expressed in the language of $\mathcal{T}$ (see [28]). ◀

Now we may associate with the base predicate $\sqsubseteq_\sigma$ of $\mathcal{L}$ the $\mathcal{T}$-predicate $\widetilde{\sqsubseteq}_\sigma$ defined as

$$\alpha \,\widetilde{\sqsubseteq}_\sigma\, \beta \equiv \forall n{:}\mathbb{N}.\ R_\sigma(\alpha,n) \Rightarrow R_\sigma(\beta,n)$$

and, accordingly, we have

$$\alpha \widetilde{=}_\sigma \beta \equiv \forall n{:}\mathbb{N}.\ R_\sigma(\alpha,n) \Leftrightarrow R_\sigma(\beta,n)\ .$$

For $\mathcal{L}$-predicates P we define their translation to a $\mathcal{T}$-predicate $\widetilde{P}$ by replacing $\sqsubseteq$ by $\widetilde{\sqsubseteq}$, leaving propositional connectives unchanged and replacing quantification over σ by quantification over $\mathbb{N}^\mathbb{N}$.

It remains to explain how one translates $\mathcal{L}$-terms to $\mathcal{T}$. For this purpose notice that Scott domains form a full subcategory of $\mathbf{Mod}(K_2)$, the modest sets in the (function) realizability topos over the second Kleene algebra K_2 as shown e.g. in [2]. As already mentioned above for the domain D_σ an admissible representation is provided by the restriction of $\widetilde{\mathsf{p}}_\sigma$ to $\mathbb{N}^\mathbb{N}$, the underlying set of K_2. Thus, for every PCF^{++} term $x_1{:}\sigma_1,\ldots,x_k{:}\sigma_k \vdash t : \tau$ one can find a primitive recursive neighbourhood function α_t in $\mathbb{N}^\mathbb{N}$ such that $\mathcal{L}$ proves $\forall\beta_1,\ldots,\beta_k{:}\mathbb{N}^\mathbb{N}.\, \widetilde{\mathsf{p}}_\tau(\alpha_t(\beta_1|\ldots|\beta_k)) = t[\widetilde{\mathsf{p}}_{\sigma_1}(\beta_1),\ldots,\widetilde{\mathsf{p}}_{\sigma_k}(\beta_k)/x_1,\ldots,x_k]$. Accordingly, we may translate the term t to the $\mathcal{T}$-term $\alpha_t(\beta_1|\ldots|\beta_k)$ (where juxtaposition denotes application in K_2 and $(\beta_1|\ldots|\beta_k)$ is a code for the respective k-tuple in $\mathbb{N}^\mathbb{N}$).

Thus, in summary, we have shown our first main result:

▶ **Theorem 2.** *For every sentence A of $\mathcal{L}$ there is a sentence $\widetilde{A}$ in the fragment $\mathcal{T}$ such that $\mathcal{L}$ proves $A \Leftrightarrow \widetilde{A}$. Thus $\mathcal{L}$ is complete relative to $\mathcal{T}$, the set of all true sentences of* **EL**.

3 The Effective Scott Model

For the effective Scott model [20] it should be possible to find an axiomatization $\mathcal{L}_e$ which is complete relative to the set $\mathcal{T}_e$ of all true arithmetic sentences. However, in this model the interpretation of types will not be cpo's anymore because not all directed suprema exist. For this reason we replace axioms (1-4) of subsection 2.1 by the following ones where the terms ε^σ are the same as in the respective subsection.

(1) $\forall f,g{:}\sigma\to\tau.f \sqsubseteq_{\sigma\to\tau} g \iff \big(\forall x,y:\sigma.f(x)\sqsubseteq g(y)\big)$
(2) for all x in σ the set $\{n:\mathtt{nat} \mid N(n)\wedge n\in\varepsilon^\sigma_x\}$ is r.e.
(3) for every r.e. set A if the subset $\{\varepsilon^\sigma_n \mid n\in A\}$ is directed then it has a supremum in σ
(4) the suprema of (3) are pointwise in case of function types.

The logic $\mathcal{L}_e$ of the effective Scott model is given by these four axioms and the axiom (5-30) of subsection 2.1.

The system $\mathcal{T}_e$ is the set of all true arithmetic sentences formulated in the obvious sublanguage of $\mathcal{L}_e$. We assume that $\mathcal{T}_e$ contains constants for all primitive recursive functions on natural numbers.

For reducing $\mathcal{L}_e$ to $\mathcal{T}_e$ we proceed essentially as in subsection 2.3. The main difference is that instead of the map $\mathsf{r}:(\mathtt{nat}\to\mathtt{nat})\to\mathtt{nat}\to\mathtt{nat}$ used there we consider a map $\mathsf{r}:\mathtt{nat}\to(\mathtt{nat}\to\mathtt{nat})$ which sends a code of a recursive enumeration of compact elements in type $\mathtt{nat}\to\mathtt{nat}$ to its supremum provided the elements given in the enumeration are consistent. As in subsection 2.3 we define $\widetilde{\mathsf{p}}_\sigma$ as $\mathsf{p}_\sigma\circ\mathsf{r}:\mathtt{nat}\to\sigma$ which is easily seen to be an *admissible numbering* of the computable elements of the effectively given domain $(D_\sigma,\varepsilon^\sigma)$ (see last chapter of [27]).

In analogy to Lemma 1 we have

▶ **Lemma 3.** *For every* PCF *type* σ *there is a* $\mathcal{T}_e$*-predicate* $R_\sigma(\ell,n)$ *such that*

(†) $\qquad R_\sigma(\ell,n)\Leftrightarrow\widetilde{\mathsf{p}}_\sigma(\ell)\in\varepsilon^\sigma_n$

is provable in $\mathcal{L}_e$*.*

Proof. The claim is evident for base type $\mathtt{nat}$. Suppose as induction hypothesis that we have achieved our goal for σ and τ already.

Now, completely analogously to the proof of Lemma 1 we can prove in $\mathcal{L}_e$ that

$$\widetilde{\mathsf{p}}_{\sigma\to\tau}(\ell)\in\varepsilon^{\sigma\to\tau}_{[\langle n_1,m_1\rangle,\ldots,\langle n_k,m_k\rangle]} \iff \bigwedge_{i=1}^{k} R_\tau(\Phi_{\sigma,\tau}(\ell,n_i),m_i)$$

where $\Phi_{\sigma,\tau}(\ell,n)$ is a term in $\mathcal{T}_e$ such that $\mathcal{L}_e$ proves $\mathsf{p}_\tau(\mathsf{p}_\to(\mathsf{r}(\ell))(\mathsf{e}_\sigma(\varepsilon^\sigma_n)))=\widetilde{\mathsf{p}}_\tau(\Phi_{\sigma,\tau}(\ell,n))$. The term $\Phi_{\sigma,\tau}$ exists because $\widetilde{\mathsf{p}}_\tau$ is an admissible numbering of the the computable elements of the effectively given domain D_τ.

According to the above, we may define $R_{\sigma\to\tau}$ inductively as

$$R_{\sigma\to\tau}(\ell,[\langle n_1,m_1\rangle,\ldots,\langle n_k,m_k\rangle])\equiv\bigwedge_{i=1}^{k} R_\tau(\Phi_{\sigma,\tau}(\ell,n_i),m_i)$$

which can be expressed in the language of $\mathcal{T}_e$ (see [28]). ◀

Now in analogy with subsection 2.3 we may associate with the base predicate $\sqsubseteq_\sigma$ of $\mathcal{L}_e$ the $\mathcal{T}_e$-predicate $\widetilde{\sqsubseteq}_\sigma$ defined as

$$n\,\widetilde{\sqsubseteq}_\sigma\, m\equiv\forall k{:}\mathbb{N}.\ R_\sigma(n,k)\Rightarrow R_\tau(m,k)$$

and, accordingly, we have

$$n\widetilde{=}_\sigma m\equiv\forall k{:}\mathbb{N}.\ R_\sigma(n,k)\Leftrightarrow R_\tau(m,k)$$

For $\mathcal{L}_e$-predicates P we define their translation to a $\mathcal{T}_e$-predicate $\widetilde{P}$ by replacing $\sqsubseteq$ by $\widetilde{\sqsubseteq}$, leaving propositional connectives unchanged and replacing quantification over σ by quantification over $\mathbb{N}$.

As before, it remains to say how one translates $\mathcal{L}_e$-terms to $\mathcal{T}_e$. For this purpose notice that effective Scott domains form a full subcategory of $\mathbf{Mod}(K_1)$, the modest sets in the (number) realizability topos (aka *effective topos*) over the first Kleene algebra K_1 as shown e.g. in [16]. Actually, for the effective domain D_σ an admissible numbering is provided by the restriction of $\widetilde{\mathsf{p}}_\sigma$ to $\mathbb{N}$, the underlying set of K_1. For every PCF^{++} term $x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k \vdash t : \tau$ one can find a primitive recursive function f_t such that $\mathcal{L}_e$ proves $\forall m_1, \ldots, m_k{:}\mathbb{N}.\, \widetilde{\mathsf{p}}_\tau(f_t(m_1, \ldots, m_k)) = t[\widetilde{\mathsf{p}}_{\sigma_1}(m_1), \ldots, \widetilde{\mathsf{p}}_{\sigma_k}(m_k)/x_1, \ldots, x_k]$. Accordingly, we may translate the term t to the $\mathcal{T}_e$-term $f_t(m_1, \ldots, m_k)$.

Thus, in summary, we have our second main result

▶ **Theorem 4.** *For every sentence A of $\mathcal{L}_e$ there is a sentence $\widetilde{A}$ in the fragment $\mathcal{T}_e$ such that $\mathcal{L}_e$ proves $A \Leftrightarrow \widetilde{A}$. Thus $\mathcal{L}_e$ is complete relative to $\mathcal{T}_e$, the set of all true sentences of arithmetic.*

4 Conclusions and Directions for Future Work

We have proved relative completeness of an extension of LCF axiomatising the Scott model with respect to the full theory of Baire space $\mathbb{N}^{\mathbb{N}}$. Similarly, an extension of the effective Scott model has been shown to be relative complete w.r.t. all true sentences of first order arithmetic. To the best of our knowledge these are original results.

As mentioned in the Introduction, one could now go on and attempt similar relative completeness proofs for axiomatisations of other models of PCF. For the observably sequential algorithms model [4] one can exploit the fact that it admits a universal type $\mathtt{nat}\to\mathtt{nat}$ as shown in [13] and its axiomatization could follow the ideas on Locally Boolean Domains presented in [10].

For fully abstract models of PCF, unfortunately, the methods of our paper cannot be applied. From Prop. 7.6 of [11] it follows that the fully abstract games model for PCF does not admit a universal type. This, however, does not entail that there does not exist a universal type in the model $\mathcal{F}$ which is obtained from the games model by taking the quotient modulo observational equivalence. The reason is that the quotient may create new retractions in $\mathcal{F}$. But even if there existed a universal PCF type in $\mathcal{F}$ there would still remain the problem that by Loader's result [12] the decisive predicates on compact elements are not effective and thus it would not be obvious how to axiomatise them.

In any case, the model $\mathcal{F}$ is particularly ill-behaved as shown in [18]. Firstly, not all functionals in the model preserve suprema of ω-chains. Secondly, many "finitary" objects are not compact in the sense of domain theory. However, the situation is much more satisfactory when looking at $\mathcal{F}$ from the point of view of "operationally based domain theory" as in [7, 25] where instead of order-theoretic suprema of ascending chains one considers limits of so-called "ω-chains" (cf. [19, 24]). Thus, it may be worthwhile to axiomatize $\mathcal{F}$ despite the problems discussed above. For instance, one could try to formulate the games model of PCF within the fragment **EL** (which allows one to speak about arenas and strategies which can be represented by functions on $\mathbb{N}$) and to consider the logical relation between the games model and $\mathcal{F}$ itself specifying which elements are realized by which strategies. This way one obtains representations of PCF types which, though different from the ones considered in this paper, can still be used for reformulating correctness assertions in terms of **EL**.

Another open problem is to find relatively complete logics for languages with higher order store as in [1]. In [11] a language λ_{co} has been exhibited that is universal for a games model $\mathcal{G}_!$ where $\mathcal{G}$ is the category of affine sequential algorithms on Curien-Lamarche games and ! is the *repetetive* exponential on $\mathcal{G}$. The reason for this universality is that there is a simple

universal type $\mathtt{nat}^{\mathtt{nat}} \to \mathtt{nat}^{\mathtt{nat}}$ whose computable elements can all be denoted by λ_{co} terms. Alas, the model $\mathcal{G}_!$ is more restrictive than the games model considered in [1] and thus we do not know whether the latter contains a universal type. At first sight the paper [9] seems to address this problem but the assertion language used there is too strong in the sense that it refers to higher order objects and thus contains already all correctness assertions. The main achievement of *loc.cit.* is rather that the program logic characterises programs up to observational equivalence.

Moreover, program specifications in *loc.cit.* have to be formulated as Hoare triples which cannot be combined by logical connectives and quantifiers. This, however, would be desirable since it allows one to avoid the problems with verification of higher-order local procedures as described in [5].

Acknowledgements This research has been supported by the EPSRC Research Grant "Relative Completeness for Logics of Functional Programs" (EPSRC EP/I01456X/1). We would like to thank Martin Berger for discussions on the completeness results of [9] and the anonymous referees for their suggestions and comments.

References

1 S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In Proc. of the 13th Symp. on Logic in Computer Science, pp. 334–344, IEEE Press, Washington, 1997.

2 I. Battenfeld, M. Schröder, and A. Simpson. A Convenient Category of Domains, Electronic Notes in Theoretical Computer Science, vol. 172, pp. 69–99, 2007.

3 A. Bauer. Realizability as Connection between Constructive and Computable Mathematics. Proc. of CCA 2005 - Second International Conference on Computability and Complexity in Analysis, pp. 378–379, 2005. Long version electronically available from `math.andrej.com/data/c2c.pdf`.

4 R. Cartwright, P.-L. Curien, and M. Felleisen. Fully abstract semantics for observably sequential languages. Inf. Comput., 111(2):297–401, 1994.

5 E.M. Clarke. Programming Language Constructs for Which it is Impossible to Obtain Good Hoare-like Axiom Systems. Journal of the Association for Computing Machinery, Vol. 26, No. l, pp. 129-147, 1979.

6 S. Cook. Soundness and completeness of an axiom system for program verification. SIAM Journal on Computing 7:70–90, 1978.

7 M. Escardó and W. Kin Ho. Operational domain theory and topology of sequential programming languages, Inf. Comput. 207(3): 411-437, 2009.

8 C.A.R. Hoare. An axiomatic basis for computer programming. Comm. ACM 12:576–583, 1969.

9 K. Honda, N. Yoshida, and M. Berger. An Observationally Complete Program Logic for Imperative Higher Order Functions. In Proc. of the 20th Symp. on Logic in Computer Science, pp. 270–279, IEEE Press, Washington, 2005.

10 J. Laird. Locally boolean domains. Theor. Comput. Sci., 342(1):132–148, 2005.

11 J. Laird. Functional Programs as Coroutines: A Semantic Analysis. Logical Methods in Computer Science, to appear.

12 R. Loader. Finitary PCF is not decidable. Theor. Comput. Sci. 266(1-2): 341-364, 2001.

13 J. Longley. Universal types and what they are good for. In GQ. Zhang, J. Lawson, Y.-M. Liu and M.-K. Luo (editors), Domain theory, logic and computation: Proc. of the 2nd International Symposium on Domain Theory, Semantic Structures in Computation 3, pp. 25-63, Kluwer, 2003.

14 T. Löw. Locally Boolean Domains and Universal Models for Infinitary Sequential Languages. PhD thesis, TU Darmstadt, 2006.

15 T. Löw and Th. Streicher. Universality results for models in locally boolean domains. In Z. Ésik, editor, Proc. of the 20th Int. Workshop Computer Science Logic, volume 4207 of Lecture Notes in Computer Science, pp. 456–470. Springer, 2006.

16 D.C. McCarty. Realizability and Recursive Mathematics. PhD Thesis, Oxford, 1984.

17 R. Milner. Fully Abstract Models of Typed λ-Calculi. Theor. Comput. Sci. 4, pp. 1-22,1977.

18 D. Normann and V.Yu. Sazonov. The extensional ordering of the sequential functionals. preprint, 2010.

19 J. van Oosten and A.K. Simpson. Axioms and (counter) examples in synthetic domain theory. Ann. Pure Appl. Logic, 104(1-3):233–278, 2000.

20 G. Plotkin. LCF considered as a programming language. TCS 5, pp. 223-255, 1977.

21 G. Plotkin. $\mathbb{T}^\omega$ as a universal domain. J. Comput. System Sci. 17, no. 2, pp. 209–236, 1978.

22 F. Regensburger. HOLCF: Higher Order Logic of Computable Functions, Proc. of the 8th International Workshop on Higher Order Logic Theorem Proving and Its Applications, pp. 293–307, vol. 957 of Lecture Notes in Computer Science, Springer, 1995.

23 B. Reus. Formalizing Synthetic Domain Theory. Journal of Automated Reasoning, 23:411–444, 1999.

24 B. Reus and Th. Streicher. General Synthetic Domain Theory – a logical approach. Math. Struct. in Comp. Sci., 9:177–223, 1999.

25 A. Rohr. A Universal Realizability Model for Sequential Computation. PhD Thesis, TU Darmstadt, 2002.

26 D. S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. Unpublished paper from 1969 reprinted in Böhm Festschrift, Theor. Comput. Sci. 121, No. 1-2, pp. 411–440, 1993.

27 Th. Streicher. Domain-theoretic Foundations of Functional Programming. World Scientific, 2006.

28 A. Troelstra and D. van Dalen. Constructivism in Mathematics. Two volumes, North Holland, 1988.

The Exact Hardness of Deciding Derivational and Runtime Complexity*

Andreas Schnabl[1] and Jakob Grue Simonsen[2]

1 **Institute of Computer Science, University of Innsbruck**
Technikerstr. 21a, A-6020 Innsbruck, Austria
andreas.schnabl@uibk.ac.at

2 **Department of Computer Science, University of Copenhagen (DIKU)**
Njalsgade 126-128, DK-2300 Copenhagen S, Denmark
simonsen@diku.dk

Abstract

For any class C of computable total functions satisfying some mild conditions, we prove that the following decision problems are complete for level Σ_2^0 of the arithmetical hierarchy: (A) Deciding whether a term rewriting system (TRS for short) has runtime complexity bounded by a function in C. (B) Deciding whether a TRS has derivational complexity bounded by a function in C.

In particular, the problems of deciding whether a TRS has polynomially (exponentially) bounded runtime complexity (respectively derivational complexity) are Σ_2^0-complete. This places deciding polynomial derivational or runtime complexity of TRSs at the same level in the arithmetical hierarchy as deciding nontermination or nonconfluence of TRSs. We proceed to show that the related problem of deciding for a single computable function f whether a TRS has runtime complexity bounded from above by f is Π_1^0-complete. We further prove that analysing the implicit complexity of TRSs is even more difficult: The problem of deciding whether a TRS accepts a language of terms accepted by some TRS with runtime complexity bounded by a function in C is Σ_3^0-complete.

All of our results are easily extended to the notion of minimal complexity (where the length of shortest reductions to normal form is considered) and remain valid under any computable reduction strategy. Finally, all results hold both for unrestricted TRSs and for the class of orthogonal TRSs.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes; F.2.2; F.4.1; F.4.2

Keywords and phrases Term rewriting, derivational complexity, arithmetical hierarchy

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.481

1 Introduction

Term rewriting is a simple model of non-deterministic computation that underlies much of declarative and functional programming. Term rewriting systems (TRSs for short) are finite sets of oriented equations (*rewriting rules*) on first-order terms–e.g. $c(c(x,y),z) \to c(x,c(y,z))$ for the associative law from algebra. The application of such a rule somewhere in a term–e.g. $c(2,c(c(3,3),1)) \to_{\mathcal{R}} c(2,c(3,c(3,1)))$–is the elementary step of computation in term rewriting.

In the last few years, *complexity analysis* has emerged as an important research field within the term rewriting community. Several measures of complexity have been considered

* This work was partially supported by a grant of the University of Innsbruck.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 481–495

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

for term rewriting. The conceptually simplest one was suggested by Hofbauer and Lautemann in [18]: The *derivational complexity function* with respect to a terminating TRS $\mathcal{R}$ relates the worst-case number of rewriting steps in a computation to the size of the initial term. Other such measures include *runtime complexity* [16], which is a restriction of derivational complexity only considering initial terms corresponding to function calls, and the complexity of the function *computed* by a TRS [8], its *implicit complexity*. While all of these measures consider a kind of worst case complexity, some average case analysis has been done, as well [7]. The focus has been on measures that take the worst case number of rewriting steps in a reduction as their metric. The most common way to establish upper bounds on these has been to infer them from a termination proof of the given TRS, see [17, 22, 30, 29] for some examples of this approach. A possible modification of this approach is to restrict existing termination proof techniques in order to obtain smaller complexity bounds; instances of this idea can be found in [19, 2, 16].

The problem of deciding, for some TRS $\mathcal{R}$, whether its derivational or runtime complexity is bounded by a single function, or a family of functions, is–unsurprisingly–undecidable. In the work reported in this paper, we follow recent investigations into the exact level of undecidability (in the arithmetical hierarchy) of questions in rewriting [24, 9], according to which many of the standard properties of rewriting (termination, normalisation, confluence) are known to be Π^0_2-complete. We also follow a much older set of investigations into the exact level of undecidability of intensional properties of programs [12, 23, 20, 1]. Compared to other models of computation such as Turing Machines, term rewriting operates in a quite nonstandard way, and it is a priori not clear that the classic results can be transferred to term rewriting. For instance, for nonlinear rewriting rules (where a variable may occur more than once in either the left- or the right-hand side) such as $f(x, x, y) \to g(x, y, y)$, it is assumed that both the equality check implicit in determining whether the rule can be applied (e.g. the first two arguments of f must be identical terms), and the copying of arbitrarily large terms (e.g. the term substituted for y can be large) can be done within a single computation step. Even more pertinent, the set of allowed "starting configurations" in derivational and runtime complexity analysis is defined much more liberally than in other models of computation. For Turing machines, only (a certain subset of all) well-formed configurations are considered, and in pure functional programs, the arguments of a function are always well-formed elements of a data type, e.g. $f(s(s(0)))$ — "f applied to 2". In contrast, derivational complexity, for example, must also consider "junk" terms that do not correspond to well-formed starting configurations such as $f(s(f(f(s(0)))))$. We verify that despite these obstacles, the classical results hold for TRSs.

Our results show that the decision problems for complexity functions such as the above are either Π^0_1-complete (for a specific function as an upper bound) or Σ^0_2-complete (for existence of an upper bound in a family of functions satisfying some mild conditions). This is in line with classical results on the degrees of undecidability of intensional complexity of programs. Our results run counter to the intuition given by the traditional approach to complexity analysis of rewriting: Upper bounds on the derivational complexity of a TRS are established by extraction from a proof of termination of the TRS. However, the exact degree of undecidability for deciding whether the derivational complexity of a TRS is (for instance) polynomially bounded is the same as for deciding whether it is *non*-terminating.

All results easily carry over to a different flavour of complexity from formal language theory, confusingly also called derivational complexity, but which is starkly different from the homonymous notion in term rewriting. In formal language theory, the derivational complexity relates an integer n to the maximum length of *shortest* derivations of sentences of length at

most n [6, 27].

For implicit complexity of TRSs, where the computational complexity of the mathematical function *computed* by a given TRS $\mathcal{R}$ is considered [5, 19, 21], the pertinent decision problems are even harder: Deciding whether the implicit complexity of $\mathcal{R}$ is, for instance, polynomial or exponential is even harder than the previously mentioned problems: Σ^0_3-complete. Again, this is in line with the classical results [12, 23]. However, in practice, the additional existential quantifier (quantifying over the possibly more efficient TRS) might make it easier to establish *upper* bounds on the implicit complexity of a TRS than to establish upper bounds on its derivational or runtime complexity as there is an additional degree of freedom in constructing the proof of the upper bound.

Finally, when using TRSs to reason about functional programs, the notion of *strategy* is usually employed to make evaluation deterministic and express for instance call-by-value and call-by-name. We show that all of our results remain valid for any computable strategy.

2 Preliminaries

We presuppose basic familiarity with computability theory [25, 10] and term rewriting [4, 28], but recall central definitions and notions of rewriting and computability below. Let $\mathcal{V}$ be a countably infinite set of variables, and $\mathcal{F}$ a finite signature of function symbols with fixed arities containing at least one symbol of arity 0. The set of terms over $\mathcal{F}$ and $\mathcal{V}$ is denoted as $\mathcal{T}(\mathcal{F},\mathcal{V})$. The set of ground terms over $\mathcal{F}$ is denoted as $\mathcal{T}(\mathcal{F})$. The *root symbol* (denoted as $\mathsf{rt}(t)$) of a term t is either t itself, if $t \in \mathcal{V}$, or the symbol f, if $t = f(t_1,\ldots,t_n)$. The *size* (denoted as $|t|$) of a term t is the total number of occurrences of variables and function symbols in t. A *substitution* is a mapping $\sigma\colon \mathcal{V} \to \mathcal{T}(\mathcal{F},\mathcal{V})$, where the number of variables x such that $\sigma(x) \neq x$ is finite. We usually write $t\sigma$ instead of $\sigma(t)$ to denote the application of σ to a term t. We introduce a fresh constant $\bullet$ (the *hole*) and define a *context* C as a term (over $\mathcal{F} \cup \{\bullet\}$ and $\mathcal{V}$) containing exactly one occurrence of $\bullet$. For a term t and a context C, $C[t]$ denotes the replacement of $\bullet$ by t in C.

Let $\mathcal{R}$ be a *finite* TRS over $\mathcal{F}$ and $\mathcal{V}$, i.e. a finite set of *rewriting rules* $l \to r$ with $l, r \in \mathcal{T}(\mathcal{F},\mathcal{V})$ such that l is not a variable, and each variable in r also occurs in l. We only consider finite TRSs in this paper. A TRS $\mathcal{R}$ induces a *rewrite relation* $\to_\mathcal{R}$ as follows: $s \to_\mathcal{R} t$ if there exist a rule $l \to r$ in $\mathcal{R}$, a context C, and a substitution σ such that $s = C[l\sigma]$ and $t = C[r\sigma]$. A term s is a *normal form* of $\to_\mathcal{R}$ if there exists no term t such that $s \to_\mathcal{R} t$. A rule $l \to r$ is *left-linear* if l does not contain multiple occurrences of the same variable. A TRS is *orthogonal* if it contains no critical pairs, and all of its rules are left-linear.

The n-fold composition of $\to_\mathcal{R}$ is denoted as $\to^n_\mathcal{R}$, and we write $\to^*_\mathcal{R}$ for the reflexive and transitive closure of $\to_\mathcal{R}$. We write $s \to^!_\mathcal{R} t$ if $s \to^*_\mathcal{R} t$ and t is a normal form of $\to_\mathcal{R}$. We write $s \to^{n,!}_\mathcal{R} t$ if $s \to^n_\mathcal{R} t$ and t is a normal form. A TRS $\mathcal{R}$ is *confluent* if for all terms s,t,u such that $u \to^*_\mathcal{R} s$ and $u \to^*_\mathcal{R} t$, there exists a term v such that $s \to^*_\mathcal{R} v$ and $t \to^*_\mathcal{R} v$. It is well-known that orthogonality of a TRS implies its confluence. The *derivation tree* of a term s wrt. $\mathcal{R}$ is the following directed graph: the nodes are all terms t such that $s \to^*_\mathcal{R} t$, and there exists an edge from t to t' iff $t \to_\mathcal{R} t'$. We say that a derivation tree is *non-circular* if no path starting from the root in the tree contains the same term more than once. Observe that if s is $\mathcal{R}$-terminating, then the derivation tree of s wrt. $\mathcal{R}$ is finite and non-circular, and s is its single source node. The *derivation height* of a term s with respect to a finitely branching, terminating binary relation $\to$ is given by $\mathsf{dh}(s,\to) = \max\{n : \exists t.s \to^n t\}$, and the *derivational complexity* function of a TRS $\mathcal{R}$ is defined as $\mathsf{dc}_\mathcal{R}(n) = \max\{\mathsf{dh}(s,\to_\mathcal{R}) : |s| \leq n\}$.

A function symbol f is a *defined symbol* of $\mathcal{R}$ if there exists a rewrite rule $l \to r$ such that $\mathsf{rt}(l) = f$; otherwise, it is a *constructor* of $\mathcal{R}$. We denote the set of defined symbols of $\mathcal{R}$ as $\mathcal{D}$, while the constructors of $\mathcal{R}$ are collected in $\mathcal{C}$. A TRS $\mathcal{R}$ is a *constructor TRS* if every rule in $\mathcal{R}$ has the shape $f(l_1, \ldots, l_n) \to r$ for $f \in \mathcal{D}$ and $l_1, \ldots, l_n \in \mathcal{T}(\mathcal{C}, \mathcal{V})$. The set of basic terms is $\mathcal{B} = \{f(v_1, \ldots, v_n) : f \in \mathcal{D} \wedge v_1, \ldots, v_n \in \mathcal{T}(\mathcal{C})\}$. The *runtime complexity* function of a TRS $\mathcal{R}$ is $\mathsf{rc}_{\mathcal{R}}(n) = \max\{\mathsf{dh}(s, \to_{\mathcal{R}}) : |s| \leq n \wedge s \in \mathcal{B}\}$.

We briefly recapitulate the arithmetical hierarchy. Let $n \in \mathbb{N}$. A set $A \subseteq \mathbb{N}$ is in Σ^0_n (respectively in Π^0_n) if there is an $(n+1)$-ary decidable predicate[1] $P(x_1, \ldots, x_n, x_{n+1})$ such that A is exactly the subset of $\mathbb{N}$ for which the unary predicate $\exists x_1. \forall x_2. \cdots Q x_n. P(x_1, \ldots, x_n, x_{n+1})$ (respectively $\forall x_1. \exists x_2. \cdots Q x_n. P(x_1, \ldots, x_n, x_{n+1})$) obtains, where Q is either $\exists$ or $\forall$ depending on whether n is odd or even (respectively even or odd). We say that a set $A \subseteq \mathbb{N}$ is Σ^0_n*-hard* (respectively Π^0_n-hard) if for every set B in Σ^0_n (respectively, for every set B in Π^0_n), there exists a computable function f such that $x \in B$ iff $f(x) \in A$ for all $x \in \mathbb{N}$. A set $A \subseteq \mathbb{N}$ is Σ^0_n*-complete* (Π^0_n-complete) if it is both contained in Σ^0_n (in Π^0_n) and Σ^0_n-hard (Π^0_n-hard).

For a function f and a set $\mathcal{G}$ of functions $\mathbb{N} \to \mathbb{N}$, we say that f *is globally bounded by a function in* $\mathcal{G}$ ($f \leq \mathcal{G}$) if there exists a function $g \in \mathcal{G}$ such that for all $n \in \mathbb{N}$, we have $f(n) \leq g(n)$. The set $\mathcal{G}$ is *closed under polynomial slowdown* if, for any $g \in \mathcal{G}$ and any polynomial P over $\mathbb{N}$, we have $f \leq \mathcal{G}$ for $f(x) = P(g(x))$. In particular, the set of all polynomials over $\mathbb{N}$ is closed under polynomial slowdown. We define the set of functions $\Xi(\mathcal{G})$ as $\bigcup_{g \in \mathcal{G}, a \in \mathbb{N}, b \in \mathbb{N}} g(a \cdot n + b)$.

Observe that if $\mathcal{G}$ is the set of polynomials with non-negative integer coefficients, then $\mathcal{G} = \Xi(\mathcal{G})$, $\Xi(\mathcal{G})$ is closed under polynomial slowdown, and $f \leq \Xi(\mathcal{G})$ iff f is bounded by a polynomial. On the other hand, if $\mathcal{G} = \bigcup_{a \in \mathbb{N}} g_a$ with $g_a(x) = a^x$, then $\Xi(\mathcal{G})$ is again closed under polynomial slowdown, and $f \leq \Xi(\mathcal{G})$ iff f is bounded by an exponential function; in order to see that $\Xi(\mathcal{G})$ is closed under polynomial slowdown, let $g_c \in \mathcal{G}$, and let P be a polynomial of degree d with coefficients at most a; then $P(g_c(n)) \leq g_{c+2}(d \cdot n + (d+1) \cdot a)$.

3 Turing Machines as Rewriting Systems

We assume that the reader is familiar with the standard definitions of Turing Machines [25]. The following is the specific formalisation of Turing Machines used throughout this paper.

▶ **Definition 1.** A *(deterministic single-tape) Turing Machine* is a triple (Q, Σ, δ), where

- Q is a finite set of *states* containing at least three distinct states q_s (the *starting state*), q_a (the *accept state*), and q_r (the *reject state*).
- Σ is a finite set of *tape symbols* containing at least two distinct symbols $\square$ (the *blank symbol*) and $\vdash$ (the *left end marker*).
- δ is a function from $Q \setminus \{q_a, q_r\} \times \Sigma$ to $Q \times \Sigma \times \{L, R\}$ and is called the *transition function*. It must be defined such that for all $q \in Q \setminus \{q_a, q_r\}$, we have $\delta(q, \vdash) = (q', \vdash, R)$ for some $q' \in Q$. L represents a move to the left, and R a move to the right.

A *(deterministic dual-tape) Turing Machine* is a triple (Q, Σ, δ), identical to a single-tape Turing Machine, except that δ is a function from $Q \setminus \{q_a, q_r\} \times \Sigma \times \Sigma$ to $Q \times \Sigma \times \{L, R\} \times \Sigma \times \{L, R\}$. We assume that for all $q \in Q \setminus \{q_a, q_r\}$ and $b \in \Sigma$, we have $\delta(q, \vdash, b) = (q', \vdash, R, b', x')$ and $\delta(q, b, \vdash) = (q'', b'', x'', \vdash, R)$ for some $q', q'' \in Q$, $b', b'' \in \Sigma$, and $x', x'' \in \{L, R\}$.

▶ **Definition 2.** A *configuration* of a single-tape Turing Machine is a triple (q, w, i), where

- $q \in Q$ denotes the current state.

[1] The predicate may be chosen to be primitive recursive without changing the notions defined.

- $w = \vdash w'$ with $w' \in \Sigma^*$ denotes the current content of the tape, apart from infinitely many $\Box$ symbols to the right of w.
- $i \in \{1 \dots |w|\}$ is the current position of the scanner.

A configuration of a dual-tape Turing Machine is a quintuple (q, w, i, v, j), where

- q, w, and i have the same meaning as for single-tape Turing Machines.
- $v = \vdash v'$ with $v' \in \Sigma^*$ denotes the current content of the second tape, except for infinitely many $\Box$ symbols to the right of v.
- $j \in \{1 \dots |v|\}$ is the current position of the scanner on the second tape.

A *start configuration* of a single-tape Turing Machine is a configuration (q, w, i) such that $q = q_s$, $w = \vdash w'$ for some *input word* w', and $i = 1$. A start configuration of a dual-tape machine is a configuration (q, w, i, v, j) such that $q = q_s$, $\vdash w'$, $i = 1$, $v = \vdash$, and $j = 1$. The *size* of a configuration $\alpha = (q, w, i)$ of a single-tape machine, denoted as $|\alpha|$, is $|w|$ (here $|w|$ denotes the length of w). For dual tape machine configurations $\alpha = (q, w, i, v, j)$, we have $|\alpha| = |w| + |v|$.

Given a (single-tape or dual-tape) Turing Machine M, the notions "M moves from configuration α to configuration β in one step", "M runs for n steps on input word x", "M halts on configuration α", "M halts on input word x", "M accepts/rejects input word x", and "M accepts (exactly) language L" are defined in the usual way. We leave it to the reader to fill out the formal details.

Finally, given a (single-tape or dual-tape) Turing Machine M, we define the functions $\textsc{Time}_M$ and $\textsc{LifeTime}_M : \mathbb{N} \to \mathbb{N}$ as follows; we chose the name $\textsc{LifeTime}_M$ to reflect the close relationship to the Turing Machine *mortality problem*, which asks whether a given Turing Machine halts on all configurations:

$$\textsc{Time}_M(n) = \max\{m : M \text{ runs } m \text{ steps on input word } x \wedge |x| \leq n\}$$
$$\textsc{LifeTime}_M(n) = \max\{m : M \text{ runs } m \text{ steps on configuration } \alpha \wedge |\alpha| \leq n\}$$

If there exists any input word of length at most n (respectively, a configuration α with $|\alpha| \leq n$) on which M does not halt, then $\textsc{Time}_M(n)$ (respectively, $\textsc{LifeTime}_M(n)$) is undefined.

We now encode dual-tape Turing Machines M as TRSs $\Delta(M)$, essentially using the encoding of [28, Chapter 5], lifted to dual-tape machines. We build up the signature $\mathcal{F}$ of $\Delta(M)$ from a set of defined symbols $\mathcal{D}$ and constructor symbols $\mathcal{C}$ as follows. For each symbol $b \in \Sigma$, the set $\mathcal{C}$ contains b as a unary function symbol, as well as a nullary symbol $\triangleright$. For each state $q \in Q$, $\mathcal{D}$ contains q as a function symbol of arity 5. Additionally, $\mathcal{D}$ contains the unary symbols ok and runM. Words over Σ are translated to terms by $\phi(\epsilon) = \triangleright$, and $\phi(bw) = b(\phi(w))$, where $b \in \Sigma$ and $w \in \Sigma^*$. A configuration (q, w, i, v, j) such that $w = w_1 \dots w_n$ and $v = v_1 \dots v_m$ is encoded as the term $q(\mathsf{ok}(\triangleright), \phi(w_{i-1} \dots w_1), \phi(w_i \dots w_n), \phi(v_{j-1} \dots v_1), \phi(v_j \dots v_m))$. This way, it is easy to simulate Turing Machine computation steps by rewriting steps. For ease of notation, we often identify w and $\phi(w)$ for $w \in \Sigma^*$ during the rest of this paper. The purpose of demanding the subterm $\mathsf{ok}(\triangleright)$ to be present in the encoding of a configuration is to ensure that configurations (in particular, unreachable configurations) can not be encoded by basic terms. Therefore, in contrast to the construction in [28, Chapter 5], $\Delta(M)$ is not a constructor TRS.

▶ **Definition 3.** Let $M = (Q, \Sigma, \delta)$ be a dual-tape Turing Machine. Then the orthogonal TRS $\Delta(M)$ is defined by the rules shown in Figure 1.

Here, the rules for the transition function of the given Turing Machine are the natural lifting of the rules given in [28, Chapter 5] to our encoding of configurations. The first of the

transition function	rewrite rule (for each $q \in Q \setminus \{q_a, q_r\}$ and $a, b, c, d \in \Sigma$)
$\delta(q, b, d) = (q', b', R, d', R)$	$q(\mathsf{ok}(\triangleright), x, by, z, dw) \to q'(\mathsf{ok}(\triangleright), b'x, y, d'z, w)$
$\delta(q, b, d) = (q', b', R, d', L)$	$q(\mathsf{ok}(\triangleright), x, by, cz, dw) \to q'(\mathsf{ok}(\triangleright), b'x, y, z, cd'w)$
$\delta(q, b, \square) = (q', b', R, d', R)$	$q(\mathsf{ok}(\triangleright), x, by, z, \triangleright) \to q'(\mathsf{ok}(\triangleright), b'x, y, d'z, \triangleright)$
$\delta(q, b, d) = (q', b', L, d', R)$	$q(\mathsf{ok}(\triangleright), ax, by, z, dw) \to q'(\mathsf{ok}(\triangleright), x, ab'y, d'z, w)$
$\delta(q, b, d) = (q', b', L, d', L)$	$q(\mathsf{ok}(\triangleright), ax, by, cz, dw) \to q'(\mathsf{ok}(\triangleright), x, ab'y, z, cd'w)$
$\delta(q, b, \square) = (q', b', L, d', R)$	$q(\mathsf{ok}(\triangleright), ax, by, z, \triangleright) \to q'(\mathsf{ok}(\triangleright), x, ab'y, d'z, \triangleright)$
$\delta(q, \square, d) = (q', b', R, d', R)$	$q(\mathsf{ok}(\triangleright), x, \triangleright, z, dw) \to q'(\mathsf{ok}(\triangleright), b'x, \triangleright, d'z, w)$
$\delta(q, \square, d) = (q', b', R, d', L)$	$q(\mathsf{ok}(\triangleright), x, \triangleright, cz, dw) \to q'(\mathsf{ok}(\triangleright), b'x, \triangleright, z, cd'w)$
$\delta(q, \square, \square) = (q', b', R, d', R)$	$q(\mathsf{ok}(\triangleright), x, \triangleright, z, \triangleright) \to q'(\mathsf{ok}(\triangleright), b'x, \triangleright, d'z, \triangleright)$
	additional rules
	$\mathsf{runM}(x) \to q_s(\mathsf{ok}(\triangleright), \triangleright, \vdash(x), \triangleright, \vdash(\triangleright))$
	$q_a(\mathsf{ok}(\triangleright), x, y, z, w) \to q_a(\triangleright, x, y, z, w)$
	$q_r(\mathsf{ok}(\triangleright), x, y, z, w) \to q_r(\triangleright, x, y, z, w)$
	$\mathsf{ok}(\mathsf{ok}(\triangleright)) \to \triangleright$

Figure 1 The TRS $\Delta(M)$ defined by a dual-tape Turing Machine M

four additional rules is responsible for rewriting a basic term of the shape $\mathsf{runM}(w)$ into the encoding of a start configuration. The other three additional rules ensure that q_a, q_r, and ok are defined symbols without violating the orthogonality of the TRS. This also implies that each term in $\mathcal{T}(\mathcal{F})$ is a word over Σ.

▶ **Lemma 4.** *Let M be a Turing Machine. Then $\mathsf{rc}_{\Delta(M)}(n) = 0$ for all $n < 2$, and $\mathsf{rc}_{\Delta(M)}(n) = \textsc{Time}_M(n-2) + 2$ for all $n \geq 2$.*

Proof. First, observe that the only basic terms t which are not normal forms with respect to $\to_{\Delta(M)}$ have runM as their root symbol. Hence, we assume $\mathsf{rt}(t) = \mathsf{runM}$. Moreover, the single argument of runM must be a word over Σ. Then the only one-step reduct of t is the encoding of a starting configuration of M. Moreover, clearly $|t| \geq 2$. A straightforward argument (compare [28, Exercise 5.3.3]) reveals the following:

- Whenever s is the encoding of a configuration α of M whose current state is not q_a or q_r, and $s \to_{\Delta(M)} s'$, then s' is the encoding of a configuration β of M such that M moves from α to β in a single step.
- Whenever s is the encoding of a configuration α of M whose current state is q_a or q_r, then $\mathsf{dh}(s, \to_{\Delta(M)}) = 1$.
- Whenever α and β are configurations of M such that M moves from α to β in a single step, then for each term encoding s of α, there exists a term encoding s' of β such that $s \to_{\Delta(M)} s'$.

From these three observations, the lemma follows immediately. ◀

4 Hardness of Runtime Complexity Analysis

We now consider the hardness of establishing upper bounds on the runtime complexity of TRSs. Thanks to Lemma 4, it is possible to use existing results about the time complexity of Turing Machines [12] directly. Note that it is necessary to use dual-tape machines in order to obtain the following results exactly as formulated. If we used single-tape machines instead,

there would be an additional quadratic slowdown in the proofs for both of the results from [12] we use. Allowing multi-tape Turing Machines instead of dual-tape machines would yield an additional speedup in the order of $n \log n$ [14]; however, this additional speedup is not needed for obtaining the results below.

▶ **Lemma 5.** *Let $\mathcal{G} = \{g_1, g_2, \ldots\}$ be a recursively enumerable set of computable, strictly increasing, and total functions $\mathbb{N} \to \mathbb{N}$. Then there is a computable, strictly increasing, and total function $f\colon \mathbb{N} \to \mathbb{N}$ such that $f \not\leq \Xi(\mathcal{G})$.*

Proof. Let $f(n) = 1 + \max\{g_1(n^2+n), \ldots, g_n(n^2+n)\}$. Then f is obviously computable, strictly increasing, and total, and for all $c, d, k \in \mathbb{N}$, we have $f(n) > g_k(c \cdot n + d)$ for all $n > \max\{c, d, k\}$. ◀

▶ **Proposition 6.** *Let $\mathcal{G}$ be a recursively enumerable set of computable, strictly increasing, and total functions $\mathbb{N} \to \mathbb{N}$. Then the following decision problem is Σ^0_2-hard:*
Instance: *A dual-tape Turing Machine M.*
Question: *Is $\textsc{Time}_M \leq \Xi(\mathcal{G})$?*

Proof. By [12, Theorem 2], the proposition holds for the special case that $\mathcal{G}$ is the set of polynomials. Inspection of the proof of [12, Theorem 2] yields that only the following two properties of $\mathcal{G}$ are used: $\Xi(\mathcal{G})$ must contain the function $n+1$, and there must exist a computable total function f such that $f \not\leq \Xi(\mathcal{G})$. The first property follows from the assumption that $\mathcal{G}$ must consist of strictly increasing total functions, and the second property follows from Lemma 5. ◀

▶ **Theorem 7.** *Let $\mathcal{G}$ be a recursively enumerable set of computable, strictly increasing, and total functions $\mathbb{N} \to \mathbb{N}$. Then the following decision problem is Σ^0_2-complete:*
Instance: *A TRS $\mathcal{R}$.*
Question: *Is $\mathsf{rc}_{\mathcal{R}} \leq \Xi(\mathcal{G})$?*
If the problem instances are restricted to orthogonal TRSs, Σ^0_2-completeness holds, as well.

Proof. To see that the problem is contained in Σ^0_2, let $P(x_1, x_2, x_3)$ be the ternary predicate on $\mathbb{N}$ that obtains exactly if the i^{th} function g_i in $\mathcal{G}$ and the TRS $\mathcal{R}$ encoded by x_3 satisfy $\mathsf{rc}_{\mathcal{R}}(x_2) \leq g_i(j \cdot x_2 + k)$, where (i, j, k) is the triple encoded by x_1. Observe that $P(x_1, x_2, x_3)$ is a decidable predicate: as $\mathcal{G}$ is recursively enumerable and consists of computable functions, we may compute $g_i(j \cdot x_2 + k)$; as the signature and set of rules of $\mathcal{R}$ are both finite, we may compute the finite set of basic terms of size at most x_2, and for each of these compute their derivation trees up to depth $g_i(j \cdot x_2 + k)$ and subsequently check whether the leaves of each tree consist only of normal forms, and whether all trees are non-circular. Thus, the answer to the question to be decided is "yes" for the TRS encoded by x_3 iff the predicate $\exists x_1 . \forall x_2 . P(x_1, x_2, x_3)$ obtains, proving containment in Σ^0_2.

We now show Σ^0_2-hardness of the problem. By Proposition 6, it is Σ^0_2-hard to decide whether $\textsc{Time}_M \leq \Xi(\mathcal{G})$, given a dual-tape Turing Machine M. From Lemma 4, it follows that $\mathsf{rc}_{\Delta(M)} \leq \Xi(\mathcal{G})$ iff $\textsc{Time}_M \leq \Xi(\mathcal{G})$. The transformation Δ is obviously computable, and $\Delta(M)$ is orthogonal. Therefore, it is Σ^0_2-hard to decide whether $\mathsf{rc}_{\mathcal{R}} \leq \Xi(\mathcal{G})$, given a TRS $\mathcal{R}$ (independent of whether $\mathcal{R}$ is restricted to be orthogonal). ◀

▶ **Proposition 8.** *Let f be a computable and total function $\mathbb{N} \to \mathbb{N}$ such that $f(n) > n$ for all $n \in \mathbb{N}$. Then the following decision problem is Π^0_1-hard:*
Instance: *A dual-tape Turing Machine M.*
Question: *Is $\textsc{Time}_M(n) \leq f(n)$ for all $n \in \mathbb{N}$?*

Proof. Straightforward generalisation of [12, Theorem 1]. ◀

▶ **Theorem 9.** *Let f be a computable and total function $\mathbb{N} \to \mathbb{N}$ such that $f(n) > n$ for all $n \in \mathbb{N}$. Then the following decision problem is Π^0_1-complete:*
Instance: *A TRS $\mathcal{R}$.*
Question: *Is $\mathsf{rc}_\mathcal{R}(n) \leq f(n)$ for all $n \in \mathbb{N}$?*
If the problem instances are restricted to orthogonal TRSs, Π^0_1-completeness holds, as well.

Proof. To see that the problem is contained in Π^0_1, consider the binary predicate $P(x_1, x_2)$ on $\mathbb{N}$ that obtains iff $\mathsf{rc}_\mathcal{R}(x_1) \leq f(x_1)$ where $\mathcal{R}$ is the TRS encoded by the integer x_2. As f is computable and total, and as the derivation tree of each of the finite number of terms of size at most x_1 can be computed up to depth $f(x_1)$, the predicate is obviously decidable. Hence, the answer to the question to be decided is "yes" iff the predicate $\forall x_1.P(x_1, x_2)$ obtains, and containment in Π^0_1 is shown.

We now show Π^0_1-hardness of the problem. Let $f'(n) = f(n+2) - 2$, and note that $f'(n) > n$. By Proposition 8, it is Π^0_1-hard to decide whether $\textsc{Time}_M(n) \leq f'(n)$ for all $n \in \mathbb{N}$, given a dual-tape Turing Machine M. By Lemma 4, we have $\mathsf{rc}_{\Delta(M)}(n) \leq f(n)$ for all $n \in \mathbb{N}$ iff $\textsc{Time}_M(n) \leq f'(n)$ for all $n \in \mathbb{N}$. The transformation Δ is obviously computable, and $\Delta(M)$ is orthogonal. Therefore, it is Π^0_1-hard to decide whether $\mathsf{rc}_\mathcal{R}(n) \leq f(n)$ for all $n \in \mathbb{N}$, given a TRS $\mathcal{R}$ (independent of whether $\mathcal{R}$ is restricted to be orthogonal). ◀

5 Implicit Computational Complexity Analysis for Rewriting

In this section we establish Σ^0_3-completeness of deciding implicit complexity bounds on TRSs: Deciding whether the computation carried out by a TRS can be done within a certain time bound, possibly by another, more efficient TRS. In the literature, there exist similar results about Turing Machines [12, 23]. In order to be able to apply them, we need to establish a link between computations carried out by TRSs and Turing Machines. For one direction of this link, Lemma 4 suffices. The existence of the other direction of the link has recently been shown by Avanzini and Moser [3]. In the following, we define a simple notion of computation by a TRS, and glue the above components together.

▶ **Definition 10.** Let $\mathcal{R}$ be a TRS with signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$, let f be a specific n-ary function symbol in $\mathcal{D}$ (we call f the *main function* of $\mathcal{R}$), and a another specific symbol in the signature of $\mathcal{F}$ (we call a the *accepting symbol* of $\mathcal{R}$). Then for $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C})$ we say that $\mathcal{R}$ *accepts* $(t_1, \ldots, t_n)$ if $f(t_1, \ldots, t_n) \to^!_\mathcal{R} t$ such that $\mathsf{rt}(t) = a$. The *language accepted by* $\mathcal{R}$ is the set $\mathcal{L}(\mathcal{R}) = \{(t_1, \ldots, t_n) : t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}) \wedge \mathcal{R} \text{ accepts } (t_1, \ldots, t_n)\}$.

▶ **Definition 11.** Let $\mathcal{R}$ be a TRS with main function f of arity n, accepting symbol a, and signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$, let $L \subseteq \mathcal{T}(\mathcal{C})^n$, and let $\mathcal{G}$ be a set of computable, strictly increasing, and total functions. We say that $\mathcal{R}$ *(deterministically) accepts L in time $\Xi(\mathcal{G})$* if $\mathcal{L}(\mathcal{R}) = L$, $\mathcal{R}$ is confluent, and $\mathsf{rc}_\mathcal{R} \leq \Xi(\mathcal{G})$.

As shown by the next lemma, Δ indeed relates the notions of acceptance for Turing Machines and TRSs in the natural way. It follows by the same arguments as Lemma 4.

▶ **Lemma 12.** *Let M be a dual-tape Turing Machine with tape alphabet Σ and accepting state q_a. For each word $x \in \Sigma^*$, M accepts x iff $\Delta(M)$ with main function runM and accepting symbol q_a accepts $\phi(x)$. Moreover, $\mathcal{L}(\Delta(M))$ is exactly the language accepted by M.*

▶ **Proposition 13.** *Let $\mathcal{G}$ be a recursively enumerable set of computable, strictly increasing, and total functions. Then the following decision problem is Σ^0_3-hard:*

Instance: *A dual-tape Turing Machine M.*
Question: *Does there exist a dual-tape Turing Machine M' accepting the same language as M such that $\textsc{Time}_{M'} \leq \Xi(\mathcal{G})$?*

Proof. By [23, Corollary 3], for each set C of decidable languages containing an infinite language A and all languages B such that $A \setminus B$ is finite, the following problem is Σ^0_3-hard:
Instance: A (dual-tape) Turing Machine M.
Question: Is the language accepted by M contained in C?
Fix C to be the set of all languages L decided by any (dual-tape) Turing Machine M' with $\textsc{Time}_{M'} \leq \Xi(\mathcal{G})$. As $\mathcal{G}$ contains a strictly increasing function, C contains an infinite language A and all languages B such that $A \setminus B$ is finite. For instance, Σ^*, where Σ is the tape alphabet of M, is a suitable instance of A here. Thus C satisfies the assumptions of [23, Corollary 3], and the proposition follows. ◀

Now we have all necessary ingredients to show the main theorem of this section. Proposition 13 yields the corresponding result for Turing Machines, while Lemma 12 and [3] form the bridge to term rewriting.

▶ **Theorem 14.** *Let $\mathcal{G}$ be a recursively enumerable set of computable, strictly increasing, and total functions such that $\Xi(\mathcal{G})$ is closed under polynomial slowdown. Then the following decision problem is Σ^0_3-complete:*
Instance: *A TRS $\mathcal{R}$.*
Question: *Does there exist a TRS which accepts $\mathcal{L}(\mathcal{R})$ in time $\Xi(\mathcal{G})$?*
If the problem instances are restricted to orthogonal TRSs, Σ^0_3-completeness holds, as well.

Proof. First we show that the problem is contained in Σ^0_3. Let $P(x_1, x_2, x_3, x_4)$ be the predicate on $\mathbb{N}$ that obtains exactly if the i^{th} function g_i in $\mathcal{G}$, the TRS $\mathcal{S}$ encoded by l, and the TRS $\mathcal{R}$ encoded by x_4 satisfy the following properties:

- x_1 encodes the 4-tuple (i, j, k, l).
- $\mathsf{rc}_{\mathcal{R}}(x_2) \leq x_3$ and $\mathsf{rc}_{\mathcal{S}}(x_2) \leq g_i(j \cdot x_2 + k)$
- $\mathcal{R}$ and $\mathcal{S}$ have the same main function f, accepting symbol a, and constructors $\mathcal{C}$ in their signatures $\mathcal{F}_{\mathcal{R}}$ and $\mathcal{F}_{\mathcal{S}}$.
- For all $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C})$ with $|f(t_1, \ldots, t_n)| \leq x_2$, there exists $u_1 \in \mathcal{T}(\mathcal{F}_{\mathcal{R}})$ with $\mathsf{rt}(u_1) = a$ and $f(t_1, \ldots, t_n) \to^!_{\mathcal{R}} u_1$ iff there exists $u_2 \in \mathcal{T}(\mathcal{F}_{\mathcal{S}})$ with $\mathsf{rt}(u_2) = a$ and $f(t_1, \ldots, t_n) \to^!_{\mathcal{S}} u_2$.

Observe that $P(x_1, x_2, x_3, x_4)$ is a decidable predicate: As $\mathcal{G}$ is recursively enumerable and consists of computable functions, we may compute $g_i(j \cdot x_2 + k)$; as the signature and set of rules of $\mathcal{R}$ (respectively $\mathcal{S}$) are both finite, we may compute the finite set of basic terms over $\mathcal{F}_{\mathcal{R}}$ (respectively $\mathcal{F}_{\mathcal{S}}$) of size at most x_2, and for each of these compute their derivation trees up to depth x_3 (respectively $g_i(j \cdot x_2 + k)$) and subsequently check whether the leaves of each tree consist only of normal forms, and whether all trees are non-circular. If that is the case, then the set of normal forms of the considered terms is finite, as well, and hence it is computable whether $f(t_1, \ldots, t_n) \to^!_{\mathcal{R}} u_1$ and $f(t_1, \ldots, t_n) \to^!_{\mathcal{S}} u_2$ for all relevant $t_1, \ldots, t_n, u_1, u_2$. As the answer to the question to be decided is "yes" for the TRS encoded by x_4 iff the predicate $\exists x_1.\forall x_2.\exists x_3.P(x_1, x_2, x_3, x_4)$ obtains, containment in Σ^0_3 is proved.

We now show Σ^0_3-hardness of the problem. By Proposition 13, it is Σ^0_3-hard to decide whether there exists a dual-tape Turing Machine M' accepting the same language as M such that $\textsc{Time}_{M'} \leq \Xi(\mathcal{G})$, given a dual-tape Turing Machine M. Let q_a be the accepting state of M. We set runM to be the main function, and q_a the accepting symbol of $\Delta(M)$. Note that $\Delta(M)$ is orthogonal, so the reduction described here works regardless of whether

the problem instance is restricted to be orthogonal. By Lemma 12, $L = \mathcal{L}(\Delta(M))$ is the language accepted by M. It remains to show that there exists a dual-tape Turing machine M' accepting L with $\textsc{Time}_{M'} \leq \Xi(\mathcal{G})$ iff there exists a TRS $\mathcal{R}'$ accepting L in time $\Xi(\mathcal{G})$.

In order to show the direction from left to right, suppose that there exists a dual-tape Turing machine M' accepting L with $\textsc{Time}_{M'} \leq \Xi(\mathcal{G})$. Then by employing Lemma 12 again, we also have $\mathcal{L}(\Delta(M')) = L$ if we set the main function to runM again, and the accepting symbol of $\Delta(M')$ to the accepting state of M'. Thus, $\Delta(M')$ (deterministically) accepts L in time $\Xi(\mathcal{G})$.

For the direction from right to left, suppose that there exists a confluent TRS $\mathcal{R}'$ with main function f, accepting symbol a, and $\mathsf{rc}_{\mathcal{R}'} \leq \Xi(\mathcal{G})$. Then by [3, Theorem 6.2] there exists a deterministic (dual-tape) Turing Machine M' such that $\textsc{Time}_{M'}(n) \in \mathsf{O}(\log(\mathsf{rc}_{\mathcal{R}'}(n))^3 \cdot \mathsf{rc}_{\mathcal{R}'}(n)^7)$. Since $\mathsf{rc}_{\mathcal{R}'} \leq \Xi(\mathcal{G})$, and $\Xi(\mathcal{G})$ is by assumption closed under polynomial slowdown, we have $\textsc{Time}_{M'} \leq \Xi(\mathcal{G})$, as well. ◀

6 Hardness of Derivational Complexity Analysis

We proceed to give the completeness result for establishing upper bounds on the derivational complexity of TRSs. Unfortunately, we cannot lift the results of Section 4 directly from runtime complexity to derivational complexity. The definition of the derivational complexity of a TRS places no restrictions on the considered starting term; in particular, we have to consider encodings of unreachable configurations in the underlying Turing Machine. The crucial ingredient of the main theorem in this section is an investigation by Herman [15] of the mortality problem for Turing Machines. Herman's proof gives a concrete reduction of the mortality problem from the halting problem that involves only a polynomial overhead in time complexity. In order to use this reduction, we switch from dual-tape to single-tape Turing Machines for this section.

▶ **Proposition 15** ([13, Theorem 6]). *Let M be a dual-tape Turing Machine. Then there exists a single-tape Turing Machine M' such that M' accepts and rejects exactly the same input words as M, and $\textsc{Time}_{M'}(n) \in \mathsf{O}(\max\{\textsc{Time}_M(n)^2, n^2\})$.*

▶ **Lemma 16.** *Let M be a single-tape Turing Machine with tape alphabet Σ. Then there exists a single-tape Turing Machine M' such that M' accepts and rejects exactly the same input words from Σ^* as M, M' halts on all configurations iff M halts on all input words, and $\textsc{LifeTime}_{M'}(n) \in \mathsf{O}(\max\{\textsc{Time}_M(n)^3, n^3\})$.*

Proof. By [15, Theorem 1], there exists a single-tape Turing Machine M' which accepts the same input words from Σ^* as M, and halts on all configurations iff M halts on all input words. The proof that $\textsc{LifeTime}_{M'}(n) \in \mathsf{O}(\max\{\textsc{Time}_M(n)^3, n^3\})$ is deferred to the extended version of this paper [26]. ◀

We now encode single-tape Turing Machines M as TRSs $\Delta_1(M)$. As in Section 3, we use the encoding of [28, Chapter 5]: a configuration (q, w, i) such that $w = w_1 \dots w_n$ is encoded as the term $q(\phi(w_{i-1} \dots w_1), \phi(w_i \dots w_n))$. However, we slightly change the rules of Δ_1 to reflect that we consider machines with only one-way infinite tapes for simplification purposes. Note that Δ_1 does not contain any mechanism to enforce any restriction on the starting term of a derivation; this is because we are considering derivational complexity (rather than runtime complexity) in this section.

▶ **Definition 17.** Let $M = (Q, \Sigma, \delta)$ be a single-tape Turing Machine. Then the orthogonal constructor TRS $\Delta_1(M)$ is defined by the rules shown in Figure 2.

transition function	rewrite rule (for each $q \in Q \setminus \{q_a, q_r\}$ and $a, b \in \Sigma$)
$\delta(q,b) = (q',b',R)$	$q(x,by) \to q'(b'x,y)$
$\delta(q,b) = (q',b',L)$	$q(ax,by) \to q'(x,ab'y)$
$\delta(q,\Box) = (q',b',R)$	$q(x,\triangleright) \to q'(b'x,\triangleright)$

Figure 2 The TRS $\Delta_1(M)$ defined by a single-tape Turing Machine M

We call a ground term of the shape $q(s,t)$ over the signature of $\Delta_1(M)$ a *restricted term* if $q \in Q$, and $s,t \in \Sigma^*$, and the first symbol of $s^{-1}t$ is $\vdash$ (here $(\cdot)^{-1}$ denotes string reversal).

▶ **Lemma 18.** *Let M be a single-tape Turing Machine. Then we have $\mathsf{dc}_{\Delta_1(M)}(n) \in \textsc{LifeTime}_M(\Omega(n))$ and $\mathsf{dc}_{\Delta_1(M)}(n) \in n \cdot \textsc{LifeTime}_M(\mathsf{O}(n))$.*

Proof. The following holds by straightforward arguments (compare [28, Exercise 5.3.3]):

- For each restricted term s encoding a configuration α of M such that $s \to_{\Delta_1(M)} s'$, the term s' is also restricted, and encodes a configuration β of M. Moreover, M moves from α to β in a single step.
- Whenever α and β are configurations of M such that M moves from α to β in a single step, then for each (restricted) term encoding s of α, there exists a (restricted) term encoding s' of β such that $s \to_{\Delta_1(M)} s'$.

The above implies that for each configuration α of M, the derivation height $\mathsf{dh}(s, \to_{\Delta_1(M)})$ is exactly the number of moves that can be done from α until M halts, where s is a (restricted) term which encodes α. Therefore, $\mathsf{dc}_{\Delta_1(M)}(n) \in \textsc{LifeTime}_M(\Omega(n))$.

It remains to show that $\mathsf{dc}_{\Delta_1(M)}(n) \in n \cdot \textsc{LifeTime}_M(\mathsf{O}(n))$. It easily follows from the above observations that $\mathsf{rc}_{\Delta_1(M)}(n) \in \textsc{LifeTime}_M(\mathsf{O}(n))$. We use the construction of [11, Appendix B.2], which allows us to lift this upper bound to starting terms of arbitrary shape. We define two functions f and g. The function f maps ground terms over the signature $\mathcal{F}$ of $\Delta_1(M)$ to pairs containing a string over the tape alphabet, and a multiset of restricted terms over $\mathcal{F}$. The purpose of f (compare [11, Lemma B.5]) is to extract a number of restricted terms from a term. The helper function g ensures that the leftmost symbol on the tape of each configuration encoded by a restricted term is indeed a $\vdash$.

$$
\begin{aligned}
f(\triangleright) &= (\triangleright, \emptyset) && \\
f(a(x)) &= (a(w), \mathcal{M}) && \text{if } a \in \Sigma,\ f(x) = (w, \mathcal{M}) \\
f(q(x,y)) &= (\triangleright, \{q(g(w,v))\} \cup \mathcal{M}_1 \cup \mathcal{M}_2) && \text{if } q \in Q,\ f(x) = (w, \mathcal{M}_1),\ f(y) = (v, \mathcal{M}_2) \\
g(\triangleright, \vdash(v)) &= (\triangleright, \vdash v) && \\
g(\triangleright, v) &= (\vdash(\triangleright), v) && \\
g(\vdash(\triangleright), v) &= (\vdash(\triangleright), v) && \\
g(a(\triangleright), v) &= (a(\vdash(\triangleright)), v) && \text{if } a \in \Sigma \setminus \{\vdash\} \\
g(a(x), v) &= (a(y), z) && \text{otherwise, if } a \in \Sigma,\ (y,z) = g(x,v)
\end{aligned}
$$

By [11, Lemma B.8], we get that for every term t over $\mathcal{F}$ with $f(t) = (w, \mathcal{M})$, the inequality $\mathsf{dh}(t, \to_{\Delta_1(M)}) \leq \sum_{s \in \mathcal{M}} \mathsf{dh}(s, \to_{\Delta_1(M)})$ obtains. Moreover, $|\mathcal{M}| \leq |t|$. Hence, $\mathsf{dc}_{\Delta_1(M)}(n) \leq n \cdot \mathsf{rc}_{\Delta_1(M)}(n)$. Thus, the above observations about restricted terms suffice in order to conclude $\mathsf{dc}_{\Delta_1(M)}(n) \in n \cdot \textsc{LifeTime}_M(\mathsf{O}(n))$. ◀

We are now able to transfer Proposition 6 to derivational complexity of term rewriting. Proposition 15 and Lemma 16 take care of the the unrestrictedness of the considered starting terms, and Lemma 18 performs the actual transfer from Turing Machines to TRSs.

▶ **Theorem 19.** *Let $\mathcal{G}$ be a recursively enumerable set of computable, strictly increasing, and total functions $\mathbb{N} \to \mathbb{N}$ such that $\Xi(\mathcal{G})$ is closed under polynomial slowdown. Then the following decision problem is Σ^0_2-complete:*
Instance: *A TRS $\mathcal{R}$.*
Question: *Is $\mathsf{dc}_{\mathcal{R}} \leq \Xi(\mathcal{G})$?*
If the instances are restricted to orthogonal or constructor TRSs, Σ^0_2-completeness also holds.

Proof. The proof of containment of the problem in Σ^0_2 is identical to Theorem 7 *mutatis mutandis*, whence we only show its Σ^0_2-hardness. By Proposition 6, it is Σ^0_2-hard to decide, given a dual-tape Turing Machine M, whether $\textsc{Time}_M \leq \Xi(\mathcal{G})$. By Proposition 15 and Lemma 16, there exists a single-tape Turing machine M' such that $\textsc{LifeTime}_{M'}(n) \in \mathsf{O}(\max\{\textsc{Time}_M(n)^6, n^6\})$, and M' accepts the same language as M. As $\Xi(\mathcal{G})$ is by assumption closed under polynomial slowdown, and contains a strictly increasing function (and hence also a function dominating $i'(n) = n^6$), we have $\textsc{LifeTime}_{M'} \leq \Xi(\mathcal{G})$ iff $\textsc{Time}_M \leq \Xi(\mathcal{G})$. Moreover, by Lemma 18, we have $\mathsf{dc}_{\Delta_1(M')} \in n \cdot \textsc{LifeTime}_{M'}(\mathsf{O}(n))$. As $\Xi(\mathcal{G})$ is closed under polynomial slowdown, it follows that $\mathsf{dc}_{\Delta_1(M')} \leq \Xi(\mathcal{G})$ iff $\textsc{LifeTime}_{M'} \leq \Xi(\mathcal{G})$. The transformations used in Proposition 15 and Lemmas 16 and 18 are obviously computable, and $\Delta_1(M')$ is orthogonal. Therefore, it is Σ^0_2-hard to decide whether $\mathsf{dc}_{\mathcal{R}} \leq \Xi(\mathcal{G})$, given a TRS $\mathcal{R}$ (independent of whether $\mathcal{R}$ is restricted to be orthogonal or a constructor TRS). Note that $\mathsf{dc}_{\Delta_1(M')} \leq \Xi(\mathcal{G})$ iff $\mathsf{rc}_{\Delta_1(M')} \leq \Xi(\mathcal{G})$, hence this is also an alternative proof of the Σ^0_2-completeness of determining whether $\mathsf{rc}_{\mathcal{R}} \leq \Xi(\mathcal{G})$, which places slightly stricter assumptions on $\mathcal{G}$, but allows $\mathcal{R}$ to be restricted to constructor TRSs. ◀

7 Hardness of Minimal Complexity

The proofs in this section and Section 8 are based on the observation that the simulation of a Turing machine M by the TRS $\Delta(M)$ has exactly one redex in each term encoding a configuration of M—that is, each restricted term. Every ilk of problem we consider concerns sets of reductions to some normal form; if there is only one possible reduction starting from every restricted term, the proofs of *hardness* of the various kinds of problems we consider remain virtually identical, regardless of whether we consider minimal or maximal reductions, and regardless of reduction strategy. This crucial observation is stated in Lemma 21 below.

▶ **Definition 20.** We define the *minimal height* of a term s wrt. a finitely branching, terminating relation $\to$ by $\mathsf{mh}(s, \to) = \min\{n : \exists t. s \to^{n,!}_{\mathcal{R}} t\}$. The twin notions of *minimal derivational complexity* and *minimal runtime complexity* of a TRS $\mathcal{R}$ are then defined by:

$$\mathsf{mdc}_{\mathcal{R}}(n) = \max\{\mathsf{mh}(s, \to_{\mathcal{R}}) : |s| \leq n\} \qquad \mathsf{mrc}_{\mathcal{R}}(n) = \max\{\mathsf{mh}(s, \to_{\mathcal{R}}) : |s| \leq n \wedge s \in \mathcal{B}\}\ .$$

▶ **Lemma 21.** *Let M be a dual-tape Turing machine and let s be a term in the signature of $\Delta(M)$ containing exactly one redex. If $s \to_{\Delta(M)} t$, then t contains at most one redex.*

Proof. By assumption, the only redex of s is the one contracted by the step $s \to_{\Delta(M)} t$. Hence, t only contains redexes created by that step. As $\Delta(M)$ is left-linear, redexes can only be created if the right-hand side of the rule $l \to r$ employed in $s \to_{\Delta(M)} t$ overlaps with a left-hand side of some other rule. Write $s \to_{\Delta(M)} t$ as $C[l\sigma] \to_{\Delta(M)} C[r\sigma]$ for a suitable context C and substitution σ. Split on cases as follows:

(a) If l is on one of the forms $q(\cdots)$ or $\mathsf{runM}(x)$, inspection of the rules of $\Delta(M)$ yields that r is on the form $q'(\mathsf{ok}(\triangleright), \cdots)$. Clearly, r can only overlap with the left-hand side of a rule $l' \to r'$ if the overlap occurs at the root of l' and r. As $\Delta(M)$ is orthogonal, at most one such rule $l' \to r'$ can exist, and hence there is at most one redex in t.

(b) If $l = \mathsf{ok}(\mathsf{ok}(\triangleright))$, then $r = \triangleright$. Obviously, $\triangleright$ is a normal form on its own. By assumption, C contains no redex on its own, and $\Delta(M)$ is orthogonal. Therefore, $C[\triangleright]$ contains at most one redex. ◀

▶ **Theorem 22.** *Theorems 7, 9, and 14 all hold with the notion of* $\mathsf{rc}_{\mathcal{R}}$ *replaced by* $\mathsf{mrc}_{\mathcal{R}}$ mutatis mutandis.

Proof. Every basic term s in an orthogonal TRS (such as $\Delta(M)$ for a dual-tape Turing Machine M) contains at most one redex. For each TRS on the form $\Delta(M)$, it is therefore immediate by Lemma 21 that the minimum and maximum lengths of reduction to normal form from s are the same. Therefore, all arguments in the hardness proofs of Theorems 7, 9, and 14 remain sound if we replace $\mathsf{rc}_{\mathcal{R}}$ by $\mathsf{mrc}_{\mathcal{R}}$, so the hardness results follow.

For containment in the respective complexity classes, observe that in the proofs of Theorems 7, 9, and 14 (each of the three distinct variations of) the predicate P considers longest maximal paths in the derivation tree of terms; this can obviously be replaced by the shortest maximal paths, as required by $\mathsf{mrc}_{\mathcal{R}}$, without affecting computability of P. ◀

▶ **Theorem 23.** *Theorem 19 holds with the notion of* $\mathsf{dc}_{\mathcal{R}}$ *replaced by* $\mathsf{mdc}_{\mathcal{R}}$ mutatis mutandis.

Proof. Containment in Σ_2^0 follows in the same way as in the proof of Theorem 22.

We now show Σ_2^0-hardness. Observe that for any (single-tape) Turing Machine M, the TRS $\Delta_1(M)$ is orthogonal, right-linear, and nonerasing. Therefore, $\Delta_1(M)$ has the diamond property, and for any term t, we have $\mathsf{dh}(t, \to_{\Delta_1(M)}) = \mathsf{mh}(t, \to_{\Delta_1(M)})$. With this, the hardness result follows by arguments identical to those in the proof of Theorem 19. ◀

8 Hardness under Strategies

The results so far concern TRSs with unconstrained rewrite relation. In the modelling of programming languages, it is common to consider TRSs with strategies dictating the redex to be contracted in each term. Using the same ideas as in the last section, the previous results in the paper carry over to the setting of TRSs with strategies[2]. Thus, the results of the previous sections of the paper remain valid under, for example, any innermost strategy, and under deterministic strategies such as the leftmost-outermost strategy.

▶ **Definition 24.** Let $\mathcal{R}$ be a TRS. A *strategy* $\mathbb{S}$ for $\mathcal{R}$ is defined by a relation $\to_{\mathbb{S}} \subseteq \to_{\mathcal{R}}$ such that any term t is a normal form of $\to_{\mathcal{R}}$ iff it is a normal form of $\to_{\mathbb{S}}$. We call a strategy for $\mathcal{R}$ *computable* if, given a term t, the (finite) set $\{t' : t \to_{\mathbb{S}} t'\}$ is computable.

The notions of runtime and derivational complexity of TRSs with strategies are defined *mutatis mutandis*. For the next theorem, Lemma 21 is again the crucial proof ingredient.

▶ **Theorem 25.** *Let f be a computable mapping returning a computable strategy $f(\mathcal{R})$ for each TRS $\mathcal{R}$. Theorems 7, 9, and 14, 19, 22 and 23 all hold for the rewrite relation of $\mathcal{R}$ with strategy $\mathbb{S} = f(\mathcal{R})$ (where the instance in each decision problem is $\mathcal{R}$).*

[2] Here we use the notion "strategy" according to [28, Definition 9.1.1]. Note that this does not cover everything that is commonly called a "strategy" in term rewriting. For instance, the proofs of this section can not be directly carried over to *context-sensitive rewriting*.

Proof. Observe that if term s contains exactly one redex, then for any term t and strategy $\mathbb{S}$ for $\mathcal{R}$, we have $s \to_{\mathcal{R}} t$ iff $s \to_{\mathbb{S}} t$. For every TRS on the form $\Delta(M)$, each basic term of $\Delta(M)$ has at most one redex. By Lemma 21, it is immediate that the lengths of all reductions to normal form from s are the same. Therefore, all arguments in the hardness proofs of Theorems 7, 9, 14, and 22, remain sound under $\mathbb{S}$.

For Σ^0_2-hardness of the remaining two properties, observe that for any (single-tape) Turing Machine M, the TRS $\Delta_1(M)$ is orthogonal, right-linear, and nonerasing. Therefore, $\Delta_1(M)$ has the diamond property, and for any term t, all reductions from t to its (unique) normal form have the same length. In particular, we have $\mathsf{dh}(t, \to_{\Delta_1(M)}) = \mathsf{dh}(t, \to_{\mathbb{S}})$. Hence, the hardness proofs for Theorems 19 and 23 remain sound when restricted to $\mathbb{S}$.

To prove containment in the respective classes of the arithmetical hierarchy, observe that each containment proof in Theorems 7, 9, 14, 19, 22, and 23 is done by computing the derivation tree starting from a term s to a certain depth. The derivation tree with respect to a strategy can be obtained by pruning the full derivation tree: A branch $t \to t''$ (and thus, the entire subtree starting from t'') is cut off if $t'' \notin \{t' : t \to_{\mathbb{S}} t'\}$. As the strategy is computable, the pruning operation is clearly computable, hence also the pruned derivation trees, and we may thus replace the trees in the proofs of the above theorems by their pruned versions, concluding the proof. ◀

9 Conclusion and Suggestions for Future Work

We have proved that a number of problems related to bounding the derivational and runtime complexity of rewrite systems are complete for classes in the arithmetical hierarchy. We hope that our results may be used to prove the exact hardness other problems in applied logic—this would avoid the tedium of pure reduction from Turing machines.

A related open problem is Problem #107 of RTALooP[3], a list of open problems collected by term rewriters: what are complete characterisations of polynomial derivational complexity?

Furthermore, recent efforts have been made to devise automated methods for showing whether the derivational or runtime complexity of a given TRS is polynomial, see for instance [2, 16, 30, 29]. All of this recent work was focused on proving termination of a TRS by some restricted means, and then extracting a complexity bound from that termination proof. However, the position of this problem in the arithmetical hierarchy (which is the same as the position of *non*termination analysis) suggests that it would be promising to try to certify polynomial complexity bounds for rewrite systems in a completely novel way.

References

1 A. Asperti. The intensional content of Rice's theorem. In *Proc. 35th POPL*, pages 113–119. ACM, 2008.

2 M. Avanzini and G. Moser. Complexity analysis by rewriting. In *Proc. 9th FLOPS*, volume 4989 of *LNCS*, pages 130–146, 2008.

3 M. Avanzini and G. Moser. Closing the gap between runtime complexity and polytime computability. In *Proc. 21st RTA*, volume 6 of *LIPIcs*, pages 33–48, 2010.

4 F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

5 G. Bonfante, E. A. Cichon, J.-Y. Marion, and H. Touzet. Algorithms with polynomial interpretation termination proof. *J. Funct. Program.*, 11(1):33–53, 2001.

[3] `http://rtaloop.mancoosi.univ-paris-diderot.fr/`

6 R. Book. Time-bounded grammars and their languages. *J. Comput. Syst. Sci.*, 5(4):397–429, 1971.

7 C. Choppy, S. Kaplan, and M. Soria. Complexity analysis of term-rewriting systems. *Theor. Comput. Sci.*, 67(2):261–282, 1989.

8 E. A. Cichon and P. Lescanne. Polynomial interpretations and the complexity of algorithms. In *Proc. 11th CADE*, volume 607 of *LNCS*, pages 139–147, 1992.

9 J. Endrullis, H. Geuvers, J. G. Simonsen, and H. Zantema. Levels of undecidability in rewriting. *Inform. Comput.*, 209(2):227 – 245, 2011.

10 M. Fernandez. *Models of Computation: An Introduction to Computability Theory*. Undergraduate topics in computer science. Springer London, 2009.

11 M. C. F. Ferreira. *Termination of term rewriting*. PhD thesis, Universiteit Utrecht, 1995.

12 P. Hájek. Arithmetical hierarchy and complexity of computation. *Theor. Comput. Sci.*, 8:227–237, 1979.

13 J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *T. Am. Math. Soc.*, 117:285–306, 1965.

14 F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape Turing Machines. *J. ACM*, 13(4):533–546, 1966.

15 G. T. Herman. Strong computability and variants of the uniform halting problem. *Math. Logic Quart.*, 17:115–131, 1971.

16 N. Hirokawa and G. Moser. Automated complexity analysis based on the dependency pair method. In *Proc. 4th IJCAR*, volume 5195 of *LNCS*, pages 364–379, 2008.

17 D. Hofbauer. Termination proofs by multiset path orderings imply primitive recursive derivation lengths. *Theor. Comput. Sci.*, 105(1):129–140, 1992.

18 D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In *Proc. 3rd RTA*, volume 355 of *LNCS*, pages 167–177, 1989.

19 J.-Y. Marion. Analysing the implicit complexity of programs. *Inform. Comput.*, 183(1):2–18, 2003.

20 J.-Y. Marion and J.-Y. Moyen. Heap-size analysis for assembly programs, 2006. Unpublished manuscript. Available at `http://hal.archives-ouvertes.fr/docs/00/06/78/38/PDF/main.pdf`.

21 J.-Y. Marion and R. Péchoux. Sup-interpretations, a semantic method for static analysis of program resources. *ACM Trans. Comput. Log.*, 10(4), 2009.

22 G. Moser and A. Schnabl. The derivational complexity induced by the dependency pair method. In *Proc. 20th RTA*, volume 5595 of *LNCS*, pages 255–269, 2009.

23 K. W. Regan. Arithmetical degrees of index sets for complexity classes. In *Logic and Machines*, volume 171 of *LNCS*, pages 118–130, 1983.

24 G. Roşu. Equality of streams is a Π^0_2-complete problem. In *Proc. 11th ICFP*. ACM, 2006.

25 H. Rogers Jr. *Theory of Recursive Functions and Effective Computability*. The MIT Press, paperback edition, 1987.

26 A. Schnabl and J. G. Simonsen. The exact hardness of deciding derivational and runtime complexity, 2011. Extended version. Available at `http://cl-informatik.uibk.ac.at/users/aschnabl`.

27 S. Sippu. Derivational complexity of context-free grammars. *Inform. Control*, 53(1–2):52–65, 1982.

28 TeReSe. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

29 J. Waldmann. Polynomially bounded matrix interpretations. In *Proc. 21st RTA*, volume 6 of *LIPIcs*, pages 357–372, 2010.

30 H. Zankl and M. Korp. Modular complexity analysis via relative complexity. In *Proc. 21st RTA*, volume 6 of *LIPIcs*, pages 385–400, 2010.

A Category Theoretic View of Nondeterministic Recursive Program Schemes

Daniel Schwencke

Institut für Theoretische Informatik,
Technische Universität Braunschweig, Germany
http://www.tu-braunschweig.de/iti
schwencke@iti.cs.tu-bs.de

Abstract

Deterministic recursive program schemes (RPS's) have a clear category theoretic semantics presented by Ghani et al. and by Milius and Moss. Here we extend it to nondeterministic RPS's. We provide a category theoretic notion of guardedness and of solutions. Our main result is a description of the canonical greatest solution for every guarded nondeterministic RPS, thereby giving a category theoretic semantics for nondeterministic RPS's. We show how our notions and results are connected to classical work.

1998 ACM Subject Classification F.3.3 Studies of Program Constructs; F.1.2 Modes of Computation

Keywords and phrases Recursive program scheme, nondeterminism, powerset monad, distributive law, final coalgebra

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.496

1 Introduction

Deterministic and nondeterministic recursive program schemes (RPS's) were investigated in the 1970's and 80's by several authors (see related work below). Different semantics for RPS's were proposed and relationships between them were proved. More recently, a category theoretic semantics for deterministic RPS's has been developed by Ghani et al. [9] and by Milius and Moss [16]. There are clear advantages of this semantics: it applies to a considerably generalized notion of RPS and it requires less assumptions than classical semantics, which need order or metric structures.

However, no category theoretic semantics for nondeterministic RPS's has been presented so far. The present paper bridges this gap: it provides a category theoretic notion of nondeterministic RPS, of guardedness and of solutions. As our main result, a semantics of the guarded nondeterministic RPS's is given by proving them to have a canonical greatest solution.

Technically this turns out to be a challenging task: parts of the techniques known from [9, 16] are not available in the nondeterministic case. Thus large technical parts of our work reflect the effort it takes to adjust the category theoretic methods to the nondeterministic case. Nevertheless, this pays off in the end: besides obtaining a semantics for a generalized notion of a nondeterministic RPS, our approach has a clear structure and is easily related to classical semantics of nondeterministic RPS's as well as to the deterministic category theoretic semantics of [9, 16]. Moreover, several generalizations and extensions can be considered (see future work in Section 6).

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 496–511

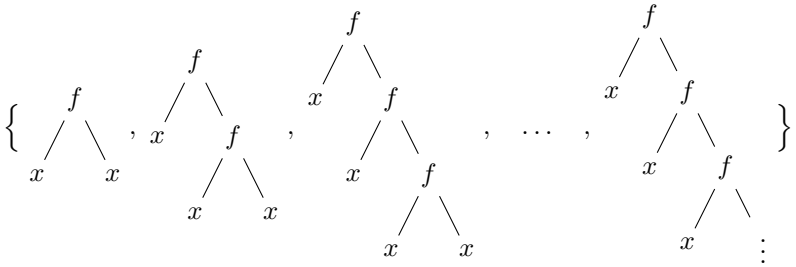
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We illustrate the topic of this paper on the nondeterministic RPS

$$\phi(x) = f(x,x) \text{ or } f(x,\phi(x))\,. \tag{1}$$

Here ϕ is a new function symbol of arity one, f a given function symbol of arity two and x a variable.

A semantics of a nondeterministic RPS attaches to every new function symbol a set of possibly infinite trees with nodes labeled by given function symbols or variables which "solves" the system of equations. In our example this is the following infinite set



(2)

of trees where the right-hand tree is infinite. Substituting every element on the right-hand side of (1) and interpreting the operation or as nondeterministic choice we get back this set. But removing the infinite tree from this set also gives such a "solution"—unlike for deterministic RPS we must make a decision when giving a semantics to nondeterministic RPS's. In our main result we prove that there always is a greatest solution that can be chosen canonically.

Structure of the Paper

Section 2 provides several notions and results needed in this paper. In Section 3 we show how to obtain a canonical distributive law of the free completely iterative monad T^H on a Set-functor H over the nonempty powerset monad $\mathcal{P}^+$. It is used in Section 4 to prove that $\mathcal{P}^+T^H$ is a weakly completely iterative monad; moreover, in this section we consider a functor $\bar{\mathcal{H}}$ on a certain Kleisli category derived from H, and show an $\bar{\mathcal{H}}$-coalgebra to be weakly final. In Section 5 we give a category theoretic notion of a nondeterministic RPS and prove our main result, namely that every guarded nondeterministic RPS has a canonical greatest solution, using the technical results from the previous section. We compare our work with an existing category theoretic notion of a deterministic RPS and with classical work on nondeterministic RPS's. Finally we give a brief summary and discuss several directions for future work in Section 6.

Nearly all proofs are omitted; they can be found in the full version on the author's web page[1].

Related Work

Different semantics of RPS's have been investigated in the 1970's and 80's: for deterministic RPS's see for example Courcelle [8], Guessarian [10] and Nivat [19]; for nondeterministic RPS's we mention Boudol [7], Arnold and Nivat [4] and Poigné [20]. In Section 5 we compare our work in particular with [4] to see how we cover the classical definitions and results. A

[1] http://www.tu-braunschweig.de/iti/mitarbeiter/ehemalige/schwencke

category theoretic approach to deterministic RPS's is given by Ghani, Lüth and de Marchi [9] and by Milius and Moss [16]. We make use of several techniques from [16]. This paper is also loosely related to our previous work [17] where we used distributive laws of the same kind to bring recursion (but on the level of equations for variables) together with effects like non-determinism.

2 Preliminaries

We assume that the reader is familiar with the basic notions of category theory such as category, functor, natural transformation and commutative diagram; we shall also need coproducts. Moreover we assume basic knowledge about algebras and coalgebras for a functor; in particular we use free algebras as well as (weakly) final coalgebras.

Monads and Distributive Laws

▶ **Definition 2.1.** A *monad* (M, η, μ) on a category $\mathcal{A}$ is an endofunctor $M : \mathcal{A} \to \mathcal{A}$ together with natural transformations $\eta : \mathrm{Id} \to M$ (called the *unit* of the monad) and $\mu : MM \to M$ (called the *multiplication* of the monad) such that the *unit laws* $\mu \cdot \eta M = \mu \cdot M\eta = \mathrm{id}$ and the *multiplication law* $\mu \cdot M\mu = \mu \cdot \mu M$ hold.
A *monad morphism* between monads (M, η^M, μ^M) and (N, η^N, μ^N) on $\mathcal{A}$ is a natural transformation $\theta : M \to N$ such that $\theta \cdot \eta^M = \eta^N$ and $\theta \cdot \mu^M = \mu^N \cdot N\theta \cdot \theta M$.

▶ **Example 2.2.** The most important example of a monad in this paper is the *nonempty powerset monad* $(\mathcal{P}^+, \eta^+, \mu^+)$ on the category Set of sets and functions:

- the functor $\mathcal{P}^+ : \mathsf{Set} \to \mathsf{Set}$ assigns to a set X the set of all nonempty subsets of X; on maps $f : X \to Y$ it is defined by $(\mathcal{P}^+ f)(X') = f[X']$ where $X' \in \mathcal{P}^+ X$;
- the X-component of the unit $\eta^+ : \mathrm{Id} \to \mathcal{P}^+$ assigns to an element $x \in X$ the singleton set $\{x\} \in \mathcal{P}^+ X$;
- the X-component of the multiplication $\mu^+ : \mathcal{P}^+\mathcal{P}^+ \to \mathcal{P}^+$ performs the union of subsets of X.

▶ **Definition 2.3.** A free monad on an endofunctor H on a category $\mathcal{A}$ is a monad (F^H, η^H, μ^H) together with a natural transformation $\kappa^H : H \to F^H$ such that for every monad (M, η^M, μ^M) on $\mathcal{A}$ together with a natural transformation $\alpha : H \to M$ there exists a unique monad morphism $\alpha^\# : F^H \to M$ such that $\alpha^\# \cdot \kappa^H = \alpha$.

▶ **Theorem 2.4** ([5])**.** *If for every object X of $\mathcal{A}$ the free H-algebras $\phi^H_X : HF^H X \to F^H X$ on X exist, the free monad on H is given objectwise by these algebras, and the free algebra maps form a natural transformation ϕ^H such that $\mu^H \cdot \phi^H F^H = \phi^H \cdot H\mu^H$ and $\phi^H = \mu^H \cdot \kappa^H F^H$.*

▶ **Definition 2.5.** The *Kleisli category* $\mathcal{A}_M$ of a monad (M, η, μ) on a category $\mathcal{A}$ is given as follows:

- the objects of $\mathcal{A}_M$ are the same objects as the ones of $\mathcal{A}$;
- the morphisms of $\mathcal{A}_M$ between X and Y are all morphisms $X \to MY$ from $\mathcal{A}$;
- the identity morphism on X is $\eta_X : X \to MX$;
- composition of $f : X \to MY$ and $g : Y \to MZ$ is given by

$$X \xrightarrow{f} MY \xrightarrow{Mg} MMZ \xrightarrow{\mu_Z} MZ\,.$$

Furthermore, there is a canonical inclusion functor $J : \mathcal{A} \to \mathcal{A}_M$ given as the identity on objects and by $Jf = \eta_Y \cdot f : X \to MY$ on morphisms $f : X \to Y$.

▶ **Definition 2.6.** A *distributive law of a functor H over a monad (M, η^M, μ^M)* is a natural transformation $\lambda : HM \to MH$ such that $\lambda \cdot H\eta^M = \eta^M H$ and $\lambda \cdot H\mu^M = \mu^M H \cdot M\lambda \cdot \lambda M$. A *distributive law of monads N and M* is a natural transformation $\lambda : NM \to MN$ such that in addition to the two laws for a distributive law of a functor over a monad (with H replaced by N) the laws $\lambda \cdot \eta^N M = M\eta^N$ and $\lambda \cdot \mu^N M = M\mu^N \cdot \lambda N \cdot N\lambda$ hold.

Throughout the paper, we denote parallel composition $G\alpha' \cdot \alpha F' = \alpha G' \cdot F\alpha' : FF' \to GG'$ of natural transformations $\alpha : F \to G$ and $\alpha' : F' \to G'$ by $\alpha * \alpha'$.

▶ **Lemma 2.7** ([6])**.** *Given a distributive law $\lambda : NM \to MN$ of monads, $(MN, \eta^M N \cdot \eta^N, (\mu^M * \mu^N) \cdot M\lambda N)$ is again a (composite) monad.*

Completely Iterative Algebras and Complete Elgot Algebras

Now (and for the rest of the paper) assume the category $\mathcal{A}$ to have binary coproducts, and let $H : \mathcal{A} \to \mathcal{A}$ be a functor. We denote coproduct injections by $\mathsf{inl} : X \to X + Y$ and $\mathsf{inr} : Y \to X + Y$ and use the notation can for the canonical morphism $[H\mathsf{inl}, H\mathsf{inr}] : HX + HY \to H(X + Y)$.

▶ **Definition 2.8.** A *flat equation morphism* in an object A (of parameters) is a morphism $e : X \to HX + A$.
A *solution* of e in an H-algebra $a : HA \to A$ is a morphism $e^\dagger : X \to A$ such that the diagram

$$\begin{array}{ccc} X & \xrightarrow{e^\dagger} & A \\ {\scriptstyle e}\downarrow & & \uparrow{\scriptstyle [a,A]} \\ HX + A & \xrightarrow[He^\dagger + A]{} & HA + A \end{array}$$

commutes.
A *completely iterative H-algebra* is an H-algebra $a : HA \to A$ in which every flat equation morphism has a unique solution.
A *complete Elgot algebra for H* is an H-algebra $a : HA \to A$ together with a function $(-)^\dagger$ assigning to each flat equation morphism e a solution $e^\dagger$ in a such that $(-)^\dagger$ is functorial and compositional (see Definition 2.10 below).

▶ **Notation 2.9.** Let $e : X \to HX + Y$ and $g : Y \to HY + A$ be flat equation morphisms and let $f : Y \to Z$ be any morphism. We denote by $f \bullet e$ the flat equation morphism $(HX + f) \cdot e : X \to HX + Z$, and we denote by $g \blacksquare e$ the flat equation morphism $(\mathsf{can} + A) \cdot (HX + g) \cdot [e, \mathsf{inr}] : X + Y \to H(X + Y) + A$.

▶ **Definition 2.10.** A function $(-)^\dagger$ assigning to each flat equation morphism e a solution $e^\dagger$ in an algebra $a : HA \to A$ is called *functorial* if for every homomorphism $h : X \to Y$ between flat equation morphisms $e : X \to HX + A$ and $g : Y \to HY + A$ (i. e. $(Hh + A) \cdot e = g \cdot h$) we have $e^\dagger = g^\dagger \cdot h$. This is, $(-)^\dagger$ is a functor between the category of all flat equation morphisms in A and their homomorphisms and the comma category of A. $(-)^\dagger$ is called *compositional* if for any equation morphisms $e : X \to HX + Y$ and $g : Y \to HY + A$ we have $(g \blacksquare e)^\dagger \cdot \mathsf{inl} = (g^\dagger \bullet e)^\dagger$.
A morphism $h : A \to B$ between complete Elgot algebras $(a : HA \to A, (-)^\dagger)$ and $(b : HB \to B, (-)^\ddagger)$ is called *solution preserving* if for all flat equation morphisms $e : X \to HX + A$ the equation $h \cdot e^\dagger = (h \bullet e)^\ddagger$ holds.

All complete Elgot algebras for H and solution preserving H-algebra homomorphisms between them form a category. It follows from [15, 2] that all completely iterative H-algebras and H-algebra homomorphisms between them form a full subcategory.

▶ **Theorem 2.11** ([15, 2]). *The following are equivalent:*

1. $\tau_X : HT^H X \to T^H X$ *is the free completely iterative* H*-algebra on* X *with universal arrow* $\eta_X : X \to T^H X$*;*
2. $\tau_X : HT^H X \to T^H X$ *is the free complete Elgot algebra for* H *on* X *with universal arrow* $\eta_X : X \to T^H X$*;*
3. $[\tau_X, \eta_X]^{-1} : T^H X \to HT^H X + X$ *is the final* $H(-) + X$*-coalgebra.*

Completely Iterative Monads

▶ **Definition 2.12.** Let (T, η, μ) be a monad on $\mathcal{A}$. A T-*module* (F, ν) is an endofunctor $F : \mathcal{A} \to \mathcal{A}$ together with a natural transformation $\nu : FT \to F$ such that the following diagrams commute:

$$\begin{array}{ccc} F & \xrightarrow{F\eta} & FT \\ & \searrow & \downarrow{\scriptstyle \nu} \\ & & F \end{array} \qquad \begin{array}{ccc} FTT & \xrightarrow{F\mu} & FT \\ {\scriptstyle \nu T}\downarrow & & \downarrow{\scriptstyle \nu} \\ FT & \xrightarrow[\nu]{} & F \end{array}$$

A *module homomorphism* between T-modules (F, ν^F) and (G, ν^G) is a natural transformation $\vartheta : F \to G$ such that $\vartheta \cdot \nu^F = \nu^G \cdot \vartheta T$.

▶ Remark 2.13. For every monad (T, η, μ), (T, μ) is a T-module.

▶ **Definition 2.14.** An *idealized monad* $(T, \eta, \mu, \bar{T}, \bar{\mu}, \vartheta)$ on $\mathcal{A}$ is a monad (T, η, μ) on $\mathcal{A}$ together with a T-module $(\bar{T}, \bar{\mu})$ and a module homomorphism $\vartheta : (\bar{T}, \bar{\mu}) \to (T, \mu)$.
An *ideal natural transformation* is a natural transformation $\alpha : F \to T$ into an idealized monad which factors

$$\alpha \equiv (\, F \xrightarrow{\bar{\alpha}} \bar{T} \xrightarrow{\vartheta} T \,).$$

An *idealized monad morphism* $(\theta, \bar{\theta})$ between idealized monads $(T, \eta^T, \mu^T, \bar{T}, \bar{\mu}^T, \vartheta^T)$ and $(S, \eta^S, \mu^S, \bar{S}, \bar{\mu}^S, \vartheta^S)$ is a monad morphism $\theta : T \to S$ together with a natural transformation $\bar{\theta} : \bar{T} \to \bar{S}$ such that the following diagrams commute:

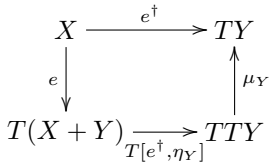
$$\begin{array}{ccc} \bar{T}T & \xrightarrow{\bar{\theta} * \theta} & \bar{S}S \\ {\scriptstyle \bar{\mu}^T}\downarrow & & \downarrow{\scriptstyle \bar{\mu}^S} \\ \bar{T} & \xrightarrow[\bar{\theta}]{} & \bar{S} \end{array} \qquad \begin{array}{ccc} \bar{T} & \xrightarrow{\bar{\theta}} & \bar{S} \\ {\scriptstyle \vartheta^T}\downarrow & & \downarrow{\scriptstyle \vartheta^S} \\ T & \xrightarrow[\theta]{} & S \end{array}$$

▶ Remark 2.15. Every monad (T, η, μ) can be canonically completed to an idealized monad $(T, \eta, \mu, T, \mu, \mathrm{id})$. In general, there are other ways to complete T to an idealized monad as we shall see in Theorem 2.17 below.

▶ **Definition 2.16.** Let $(T, \eta, \mu, \bar{T}, \bar{\mu}, \vartheta)$ be an idealized monad. An *equation morphism* is a morphism $e : X \to T(X + Y)$. It is called *guarded* if it factors

$$e \equiv (\, X \xrightarrow{e'} \bar{T}(X + Y) + Y \xrightarrow{[\vartheta_{X+Y}, \eta_{X+Y} \cdot \mathsf{inr}]} T(X + Y) \,)$$

for some e'. A *solution* of e is a morphism $e^\dagger : X \to TY$ such that the diagram



commutes. An idealized monad is called *completely iterative* if every guarded equation morphism has a unique solution. It is called *weakly completely iterative* if every guarded equation morphism has a solution.

All idealized monads on $\mathcal{A}$ together with the idealized monad morphisms form a category. In particular, we are interested in the free completely iterative monads on functors $H : \mathcal{A} \to \mathcal{A}$, albeit not in their freeness property.

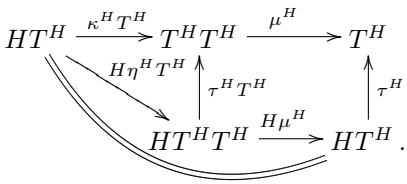
▶ **Theorem 2.17** ([15]). *Let $H : \mathcal{A} \to \mathcal{A}$ be a functor such that for every object X of $\mathcal{A}$ the final $H(-) + X$-coalgebra exists. The free completely iterative monad on H is given by $(T^H, \eta^H, \mu^H, HT^H, H\mu^H, \tau^H)$ with universal ideal natural transformation $\kappa^H : H \to T^H$ where*

- *T^H is defined on objects X as the free completely iterative H-algebra $T^H X$ on X and on morphisms $f : X \to Y$ as the unique homomorphism between the free completely iterative H-algebras on X and Y extending $\eta_Y \cdot f : X \to T^H Y$;*
- *η^H_X is given by the universal arrow of the free completely iterative H-algebra on X;*
- *μ^H_X is given as the unique homomorphism between the free completely iterative H-algebras on $T^H X$ and on X extending $\mathrm{id}_{T^H X}$;*
- *τ^H_X is given by the structure of the free completely iterative H-algebra on X; and*
- *κ^H_X is given by $\tau^H_X \cdot H\eta^H_X$.*

▶ Remark 2.18. By the definition of μ in Theorem 2.17 we have $\mu^H_X \cdot \tau^H_{T^H X} = \tau^H_X \cdot H\mu^H_X$ for every X, and from the same theorem we know that μ^H and τ^H are natural transformations. Consequently it holds $\mu^H \cdot \tau^H T^H = \tau^H \cdot H\mu^H$.

▶ **Lemma 2.19.** *It holds $\tau^H = \mu^H \cdot \kappa^H T^H$.*

Proof. Consider the diagram



The triangle is the definition of κ, the lower part is one of the monad unit laws, and for the right-hand square see Remark 2.18. Thus the desired outer square commutes. ◀

3 Canonical Distributive Laws over $\mathcal{P}^+$

In this section we provide canonical distributive laws of polynomial Set-functors H and the corresponding free completely iterative monads T^H on H over the nonempty powerset monad $\mathcal{P}+$ and prove some properties of them. These distributive laws are an integral part of our category theoretic approach to nondeterministic computations since they formalize the idea of non-determinism that all possible choices are considered.

▶ **Definition 3.1.** A *polynomial* Set*-functor* H is a Set-functor of the form $HX = \coprod_{\sigma\in\Sigma} X^{n_\sigma}$, where Σ is a signature of (possibly infinitely many) operation symbols σ with (finite) arities n_σ. We write H_Σ for the polynomial Set-functor associated with the signature Σ; elements from $H_\Sigma X$ are denoted by $\sigma(x_1, \ldots, x_n)$ where $\sigma \in \Sigma$ and $x_1, \ldots, x_n \in X$.

▶ **Lemma 3.2** ([11])**.** *There exist canonical distributive laws* $\lambda : HM \to MH$ *of every polynomial* Set*-functor* H *over every commutative monad* M *on* Set*.*

▶ Remarks 3.3. **1.** We do not state the definition of a commutative monad here, but only mention that $\mathcal{P}^+$ is commutative which is sufficient for our purposes. For more details, see Kock's papers [13, 14].

2. Our work in [17] extends Lemma 3.2 to the wider class of analytic functors; for $\mathcal{P}^+$ there even exist canonical distributive laws $\lambda : H\mathcal{P}^+ \to \mathcal{P}^+H$ for every weak pullback preserving functor H, see e. g. [12].

▶ **Example 3.4.** For every polynomial functor H_Σ, the canonical distributive law $\lambda : H_\Sigma\mathcal{P}^+ \to \mathcal{P}^+H_\Sigma$ is given by $\lambda_X(\sigma(X_1, \ldots, X_n)) = \{\sigma(x_1, \ldots, x_n) \mid x_i \in X_i, 1 \le i \le n\}$ for every n-ary operation symbol $\sigma \in \Sigma$ and $X_1, \ldots, X_n \in \mathcal{P}^+X$.

If H is a polynomial Set-functor, so is $H(-) + X$ for every set X. The final coalgebra of $H(-) + X$ is carried by the set T^HX of all finite and infinite trees with nodes labeled by operation symbols from the signature corresponding to H or by constant elements from X, where the number of children is given by the arity of the operation symbols labeling a node (see [1], Example 2.7). Whenever trees are mentioned in this paper, such trees are meant. We shall also refer to the elements of T^HX as finite and infinite terms built from operation symbols from the signature corresponding to H over variables from X.

Since for a polynomial Set-functor H final coalgebras $H(-) + X$ exist for every set X, the free completely iterative monad $(T^H, \eta^H, \mu^H, HT^H, H\mu^H, \tau^H)$ on H together with the universal natural transformation κ^H exists and is given as in Theorem 2.17. Explicitly, for a polynomial Set-functor H the natural transformations involved act as follows:

- $\eta^H_X : X \to T^HX$ considers a variable as a singleton tree;
- $\mu^H_X : T^HT^HX \to T^HX$ considers a tree with leaves labeled by trees with leaves labeled by variables from X as a tree with leaves labeled by variables from X by using the leaf labels as subtrees;
- $\tau^H_X : HT^HX \to T^HX$ acts similar as μ^H_X but for a flat tree (i. e. one of depth one) with leaves labeled by trees (of arbitrary depth); and
- $\kappa^H_X : HX \to T^HX$ considers a flat tree as a tree.

We shall leave out the superscript H when the functor H is clear from the context.
For the following proposition recall Definition 2.8 of a complete Elgot algebra.

▶ **Proposition 3.5.** *For the canonical distributive law* λ *of a polynomial* Set*-functor* H *over the monad* $\mathcal{P}^+$*,* $\mathcal{P}^+\tau_Y \cdot \lambda_{T^HY} : H\mathcal{P}^+T^HY \to \mathcal{P}^+T^HY$ *is a complete Elgot algebra for* H *for every set* Y*.*

Proof (sketch). For every set Y we define a function $(-)^\dagger$ which assigns to each flat equation morphism $e : X \to HX + \mathcal{P}^+T^HY$ a morphism $e^\dagger : X \to \mathcal{P}^+T^HY$. For example, let e be given by the system

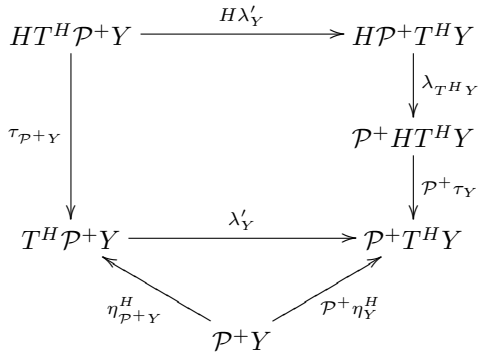
$$x_0 = \sigma(x_0, x_1)$$
$$x_1 = \{t_0, t_1\}$$

of equations where σ is an operation symbol from the signature associated with the polynomial functor H, $x_0, x_1 \in X$ and $t_0, t_1 \in T^H Y$. Then we let $e^\dagger(x_0)$ consist of the unique solutions in the free completely iterative H-algebra τ_Y on Y of all variables of all flat equation morphisms $\bar{e} : \bar{X} \to H\bar{X} + T^H Y$ which are "over" x_0: for example $\bar{x}_0$ and $\bar{x}_1$ in the system

$$\begin{aligned}\bar{x}_0 &= \sigma(\bar{x}_1, \bar{x}_2)\\ \bar{x}_1 &= \sigma(\bar{x}_0, \bar{x}_3)\\ \bar{x}_2 &= t_0\\ \bar{x}_3 &= t_1\end{aligned}$$

are "over" x_0 since there is a function $\bar{X} \to X$ mapping $\bar{x}_0$ and $\bar{x}_1$ to x_0 (and $\bar{x}_2$ and $\bar{x}_3$ to x_1) which is homomorphic for equations with right-hand sides from HX and otherwise relates variables whose right-hand sides are in the containment relation $\in$. Similarly we define $e^\dagger(x_1)$ which is easily seen to be $\{t_0, t_1\}$. Then $e^\dagger$ can be shown to be a greatest solution of e w. r. t. to subset inclusion on $\mathcal{P}^+ T^H Y$. Equivalently, $e^\dagger$ can be obtained as a greatest fixed point of an operator corresponding to the solution diagram in Definition 2.8 with $(A, a) = (\mathcal{P}^+ T^H Y, \mathcal{P}^+ \tau_Y \cdot \lambda_{T^H Y})$. This enables us to use the dual of the proof of Proposition 3.5 from [2] in order to show that $(-)^\dagger$ is functorial and compositional which concludes the current proof. ◀

▶ **Definition 3.6.** Given the canonical distributive law λ of a polynomial Set-functor H over the monad $\mathcal{P}^+$, we define for every set Y the map $\lambda'_Y : T^H \mathcal{P}^+ Y \to \mathcal{P}^+ T^H Y$ to be the unique homomorphism between the free complete Elgot algebra $\tau_{\mathcal{P}^+ Y}$ on $\mathcal{P}^+ Y$ and the complete Elgot algebra $\mathcal{P}^+ \tau_Y \cdot \lambda_{T^H Y}$ (see Proposition 3.5) extending $\mathcal{P}^+ \eta^H_Y$, i. e. λ'_Y is uniquely determined by the following commutative diagrams:



(3)

▶ **Lemma 3.7.** *The maps $\lambda'_Y : T^H \mathcal{P}^+ Y \to \mathcal{P}^+ T^H Y$ from Definition 3.6 act as follows: given a tree $t \in T^H \mathcal{P}^+ Y$ where leaves may be labeled with nonempty subsets of Y, $\lambda'_Y(t)$ is the set of all trees obtained by choosing in each of these leaves one element from the labeling set.*

▶ **Proposition 3.8.** *The canonical distributive law $\lambda : H\mathcal{P}^+ \to \mathcal{P}^+ H$ of a polynomial* Set*-functor H over the monad $\mathcal{P}^+$ extends to a distributive law $\lambda' : T^H \mathcal{P}^+ \to \mathcal{P}^+ T^H$ of monads.*

Proof (sketch). We prove that, given the canonical distributive law λ of a polynomial Set-functor H over the monad $\mathcal{P}^+$, the maps λ'_Y from Definition 3.6 form a distributive law of monads. Since one of the axioms for a distributive law of monads is already given by the lower triangle in diagram (3), we need to prove naturality of λ' and the three remaining

axioms. The proof uses freeness of the complete Elgot algebras $\tau_Y : HT^HY \to T^HY$ (see Theorem 2.11) for naturality and one of the remaining axioms, and the concrete description of λ' from Lemma 3.7 for the remaining two axioms. ◀

▶ **Lemma 3.9.** *For a distributive law λ' obtained from λ according to Proposition 3.8 we have $\lambda' \cdot \kappa\mathcal{P}^+ = \mathcal{P}^+\kappa \cdot \lambda$.*

Proof. The lemma is an easy consequence of the definitions of κ (Theorem 2.17) and λ' (diagram (3)) and therefore left to the reader. ◀

4 A Weakly Final Coalgebra

Milius and Moss ([16], Theorem 6.5) proved guarded deterministic RPS's to have unique solutions by exploiting the finality of the coalgebra $[\tau^H, \eta^H]^{-1}$ for some functor $\mathcal{H}$. As we have seen in the introduction, in the nondeterministic case solutions need not be unique. However, as our main result we shall provide in Section 5 canonical greatest solutions of nondeterministic RPS's. There we exploit weak finality of the coalgebra $J[\tau^H, \eta^H]^{-1}$ for a lifting $\bar{\mathcal{H}}$ of $\mathcal{H}$ to a suitable Kleisli category with inclusion functor J, which is proved in the present section.

▶ **Definition 4.1.** Given a distributive law $\delta : NM \to MN$ of monads, a *δ-distributive law* of an N-module $(\bar{N}, \bar{\mu}^N)$ over the monad M is a natural transformation $\bar{\delta} : \bar{N}M \to M\bar{N}$ such that the first two laws from Definition 2.6 (with H replaced by $\bar{N}$ and λ replaced by $\bar{\delta}$) and the law $\bar{\delta} \cdot \bar{\mu}^N M = M\bar{\mu}^N \cdot \bar{\delta}N \cdot \bar{N}\delta$ hold.

▶ **Lemma 4.2.** *Let $\delta : NM \to MN$ be a distributive law of the idealized monad $(N, \eta^N, \mu^N, \bar{N}, \bar{\mu}^N, \vartheta)$ over the monad (M, η^M, μ^M), and let $\bar{\delta} : \bar{N}M \to M\bar{N}$ be a δ-distributive law such that $M\vartheta \cdot \bar{\delta} = \delta \cdot \vartheta M$. Then the composite monad induced by δ is an idealized monad $(MN, \eta^M N \cdot \eta^N, (\mu^M * \mu^N) \cdot M\delta N, M\bar{N}, (\mu^M * \bar{\mu}^N) \cdot M\bar{\delta}N, M\vartheta)$.*

Specializing to $M = \mathcal{P}^+$ and $N = T^H$, we can now prove the following

▶ **Theorem 4.3.** *Let H be a polynomial* Set*-functor. For the extension $\lambda' : T^H\mathcal{P}^+ \to \mathcal{P}^+T^H$ of the canonical distributive law $\lambda : H\mathcal{P}^+ \to \mathcal{P}^+H$ (cf. Proposition 3.8),*

$$(\mathcal{P}^+T^H, \eta^+T^H \cdot \eta^H, (\mu^+ * \mu^H) \cdot \mathcal{P}^+\lambda'T^H, \mathcal{P}^+HT^H, (\mu^+ * H\mu^H) \cdot \mathcal{P}^+\lambda T^HT^H \cdot \mathcal{P}^+H\lambda'T^H, \mathcal{P}^+\tau)$$

is a weakly completely iterative monad (see Definition 2.16).

Proof (sketch). We know from Theorem 2.17 that $(T^H, \eta^H, \mu^H, HT^H, H\mu^H, \tau)$ is the free completely iterative monad on H, i. e. in particular, it is an idealized monad. Moreover, $\lambda T^H \cdot H\lambda' : HT^H\mathcal{P}^+ \to \mathcal{P}^+HT^H$ is easily seen to be a λ'-distributive law such that $\mathcal{P}^+\tau \cdot \lambda T^H \cdot H\lambda' = \lambda' \cdot \tau\mathcal{P}^+$. Thus we can apply Lemma 4.2 to see that the six-tuple in the statement of the theorem is an idealized monad. We still have to check that every guarded equation morphism has a solution. This is done by deriving deterministic guarded equation morphisms from the given nondeterministic one (similar to the proof of Proposition 3.5) and showing that the (unique) solutions of the former constitute a solution of the latter. ◀

▶ Remark 4.4. The part of the proof of Theorem 4.3 showing that all guarded equation morphisms have a solution even works for (non-guarded) equation morphisms $e : X \to \mathcal{P}^+T^H(X + Y)$ that factor

$$e = \left(X \xrightarrow{e'} \mathcal{P}^+(HX + Y) \xrightarrow{\mathcal{P}^+(\kappa_X^H + \eta_Y^H)} \mathcal{P}^+(T^HX + T^HY) \xrightarrow{\mathcal{P}^+\mathsf{can}} \mathcal{P}^+T^H(X + Y)\right).$$

The reason is that although the equation morphism e is not necessarily guarded, the derived deterministic equation morphisms always are; the rest of the proof remains the same.

Observe that for every set Y, $\mathcal{P}^+Y$ carries the partial order $\subseteq$ given by subset inclusion. This extends elementwise to a partial order $\leq$ on all sets $\mathsf{Set}(X, \mathcal{P}^+Y)$ of functions from some set X into $\mathcal{P}^+Y$, i. e. $f \leq g \Leftrightarrow \forall x \in X : f(x) \subseteq g(x)$ for functions $f, g \in \mathsf{Set}(X, \mathcal{P}^+Y)$. In this sense we use the term "greatest solution/homomorphism" in the following lemma, in Lemma 4.11 below and in Section 5.

▶ **Lemma 4.5.** *The canonical solutions $e^\dagger$ of (guarded) equation morphisms e from the proof of Theorem 4.3 and Remark 4.4 are greatest solutions; moreover, for all solutions s of a (guarded) equation morphism e the sets of all finite cuttings of the trees from $e^\dagger(x)$ and $s(x)$ are the same for every $x \in X$.*

Let us denote by $[\mathcal{A}, \mathcal{A}]$ the category of all $\mathcal{A}$-endofunctors and natural transformations between them. Any functor $H : \mathcal{A} \to \mathcal{A}$ gives rise to a functor $\mathcal{H} : [\mathcal{A}, \mathcal{A}] \to [\mathcal{A}, \mathcal{A}]$ defined on objects (i. e. functors F) and morphisms (i. e. natural transformations $\alpha : F \to G$) by $\mathcal{H}F = HF + \mathrm{Id}$ and $\mathcal{H}\alpha = H\alpha + \mathrm{id}$.
And any monad (M, η^M, μ^M) on $\mathcal{A}$ gives rise to a monad $(\mathcal{M}, \eta^{\mathcal{M}}, \mu^{\mathcal{M}})$ on $[\mathcal{A}, \mathcal{A}]$ as follows: the functor $\mathcal{M}$ is defined by $\mathcal{M}F = MF$ and $\mathcal{M}\alpha = M\alpha$, and the F-components of unit and multiplication are given by $\eta^{\mathcal{M}}_F = \eta^M F$ and $\mu^{\mathcal{M}}_F = \mu^M F$. The monad laws follow straight from the ones for (M, η^M, μ^M).

▶ **Lemma 4.6.** *Any distributive law λ of a functor H over a monad M on $\mathcal{A}$ induces a distributive law Λ of the functor $\mathcal{H}$ over the monad $\mathcal{M}$ on $[\mathcal{A}, \mathcal{A}]$.*

Proof. For every object from $[\mathcal{A}, \mathcal{A}]$ (i. e. every functor $F : \mathcal{A} \to \mathcal{A}$) we define $\Lambda_F = \mathsf{can} \cdot (\lambda F + \eta^M)$. Naturality of Λ is proved by the commutative diagram

$$\begin{array}{ccccc}
 & & \overset{\Lambda_F}{\frown} & & \\
\mathcal{H}\mathcal{M}F = HMF + \mathrm{Id} & \xrightarrow{\lambda F + \eta^M} & MHF + M & \xrightarrow{\mathsf{can}} & M(HF + \mathrm{Id}) = \mathcal{M}\mathcal{H}F \\
\downarrow{\scriptstyle \mathcal{H}\mathcal{M}\alpha = HM\alpha + \mathrm{id}} & & \downarrow{\scriptstyle MH\alpha + M\mathrm{id}} & & \downarrow{\scriptstyle M(H\alpha + \mathrm{id}) = \mathcal{M}\mathcal{H}\alpha} \\
\mathcal{H}\mathcal{M}G = HMG + \mathrm{Id} & \xrightarrow[\lambda G + \eta^M]{} & MHG + M & \xrightarrow[\mathsf{can}]{} & M(HG + \mathrm{Id}) = \mathcal{M}\mathcal{H}G \\
 & & \underset{\Lambda_G}{\smile} & &
\end{array}$$

for every morphism from $[\mathcal{A}, \mathcal{A}]$ (i. e. every natural transformation $\alpha : F \to G$ from $\mathcal{A}$): the left-hand part commutes by naturality of λ, and the right-hand part by naturality of can. The two axioms for Λ are easily checked componentwise for every object from $[\mathcal{A}, \mathcal{A}]$ (i. e. for every functor F) in

$$\Lambda_F \cdot \mathcal{H}\eta^{\mathcal{M}}_F = \mathsf{can} \cdot (\lambda F + \eta^M) \cdot (H\eta^M F + \mathrm{Id}) = \mathsf{can} \cdot (\eta^M HF + \eta^M) = \eta^M(HF + \mathrm{Id}) = \eta^{\mathcal{M}}_{\mathcal{H}F}$$

and

$$\begin{aligned}
\Lambda_F \cdot \mathcal{H}\mu^{\mathcal{M}}_F &= \mathsf{can} \cdot (\lambda F + \eta^M) \cdot (H\mu^M F + \mathrm{Id}) \\
&= \mathsf{can} \cdot (\mu^M HF + \mu^M) \cdot (M\lambda F + M\eta^M) \cdot (\lambda MF + \eta^M) \\
&= \mu^M(HF + \mathrm{Id}) \cdot M\mathsf{can} \cdot \mathsf{can} \cdot (M\lambda F + M\eta^M) \cdot (\lambda MF + \eta^M) \\
&= \mu^M(HF + \mathrm{Id}) \cdot M\mathsf{can} \cdot M(\lambda F + \eta^M) \cdot \mathsf{can} \cdot (\lambda MF + \eta^M) \\
&= \mu^{\mathcal{M}}_{\mathcal{H}F} \cdot \mathcal{M}\Lambda_F \cdot \Lambda_{\mathcal{M}F} \,.
\end{aligned}$$

◀

Now recall Definition 2.5 (Kleisli category).

▶ **Proposition 4.7** ([18]). *For any functor $H : \mathcal{A} \to \mathcal{A}$ and monad M on $\mathcal{A}$ the following are equivalent:*

1. *there is a distributive law $\lambda : HM \to MH$ of the functor over the monad;*
2. *H lifts to a functor $\bar{H}$ on $\mathcal{A}_M$.*

▶ Remark 4.8. We do not state the definition of a lifting of a functor here; let us only remark that in the proof of Proposition 4.7 for a given distributive law $\lambda : HM \to MH$ the corresponding functor $\bar{H}$ on $\mathcal{A}_M$ is given by $\bar{H}X = HX$ on objects X of $\mathcal{A}_M$ and by $\bar{H}f = \lambda_Y \cdot Hf : HX \to MHY$ on morphisms $f : X \to MY$ of $\mathcal{A}_M$.

▶ **Corollary 4.9.** *$\mathcal{H}$ lifts to a functor $\bar{\mathcal{H}}$ on $[\mathcal{A}, \mathcal{A}]_{\mathcal{M}}$.*

Explicitly $\bar{\mathcal{H}}$ is given on objects (i. e. functors F) and morphisms (i. e. natural transformations $\alpha : F \to MG$) by

$$\bar{\mathcal{H}}F = HF + \mathrm{Id} \quad \text{and} \quad \bar{\mathcal{H}}\alpha = \mathsf{can} \cdot (\lambda G + \eta^M) \cdot (H\alpha + \mathrm{id})\,.$$

Let us come back to the setting where $H : \mathsf{Set} \to \mathsf{Set}$ is polynomial, $M = \mathcal{P}^+$ and $\lambda : H\mathcal{P}^+ \to \mathcal{P}^+H$ is canonical.

▶ **Theorem 4.10.** *$J[\tau, \eta]^{-1} : T^H \to \mathcal{P}^+(HT^H + \mathrm{Id})$ is a weakly final $\bar{\mathcal{H}}$-coalgebra.*

Proof (sketch). The components of every $\bar{\mathcal{H}}$-coalgebra give rise to equation morphisms whose canonical solutions from the proof of Theorem 4.3 can be shown to form a homomorphism h into the $\bar{\mathcal{H}}$-coalgebra $J[\tau, \eta]^{-1}$. ◀

▶ **Lemma 4.11.** *The $\bar{\mathcal{H}}$-coalgebra homomorphisms $h : F \to \mathcal{P}^+T^H$ into the weakly final $\bar{\mathcal{H}}$-coalgebra from the proof of Theorem 4.10 are (componentwise) the greatest such homomorphisms; moreover, for every $\bar{\mathcal{H}}$-coalgebra homomorphism $\alpha : F \to \mathcal{P}^+T^H$ the sets of all finite cuttings of trees from $\alpha_X(z)$ and $h_X(z)$ are the same for every set X and every $z \in FX$.*

Proof. This follows from Lemma 4.5 and the proof of Theorem 4.10. ◀

5 Nondeterministic Recursive Program Schemes

In this section, we present our category theoretic notion of a nondeterministic RPS. We compare this notion with the one of a deterministic RPS from Milius and Moss [16] and with the classical notion of a nondeterministic RPS as given by Arnold and Nivat [4]. Using the technical results from the previous section, we prove our main theorem giving a semantics to nondeterministic RPS's.

▶ **Definition 5.1.** Let H and V be polynomial Set-functors. A *nondeterministic recursive program scheme* (or *NDRPS*, for short) is a natural transformation $e : V \to \mathcal{P}^+F^{H+V}$. It is called *guarded* if it factors

$$e \equiv (V \xrightarrow{e'} \mathcal{P}^+HF^{H+V} \xrightarrow{\mathcal{P}^+\mathsf{in}l F^{H+V}} \mathcal{P}^+(H+V)F^{H+V} \xrightarrow{\mathcal{P}^+\phi^{H+V}} \mathcal{P}^+F^{H+V})\,.$$

An *uninterpreted solution* of e is a natural transformation $e^\dagger : V \to \mathcal{P}^+T^H$ such that the diagram

$$\begin{array}{ccc} V & \xrightarrow{e^\dagger} & \mathcal{P}^+T^H \\ {\scriptstyle e}\downarrow & & \uparrow{\scriptstyle \mu^+T^H} \\ \mathcal{P}^+F^{H+V} & \xrightarrow[\mathcal{P}^+[\eta^+T^H \cdot \kappa^H, e^\dagger]^\#]{} & \mathcal{P}^+\mathcal{P}^+T^H \end{array} \tag{4}$$

commutes.

▶ Remark 5.2. Notice that $[\eta^+ T^H \cdot \kappa^H, e^\dagger]^\#$ in Definition 5.1 is the unique monad morphism such that $[\eta^+ T^H \cdot \kappa^H, e^\dagger]^\# \cdot \kappa^{H+V} = [\eta^+ T^H \cdot \kappa^H, e^\dagger]$. It exists since F^{H+V} is the free monad on $H + V$ with universal natural transformation $\kappa^{H+V} : H + V \to F^{H+V}$ (cf. Definition 2.3) and $\mathcal{P}^+ T^H$ is a monad by Theorem 4.3. Explicitly, for polynomial functors H and V and any set X, $F^{H+V}X$ is the set of all finite trees or terms built from the operation symbols from the signatures associated with H and V and the variables from X (similar to our description of $T^H X$ above Proposition 3.5). $[\eta^+ T^H \cdot \kappa^H, e^\dagger]^\#_X$ performs a nondeterministic variant of second-order substitution in trees (cf. [16], Section 4.1).

▶ Remark 5.3. In Definition 6.1 of [16] deterministic RPS's are defined as natural transformations $e : V \to T^{H+V}$ where H and V are endofunctors on any category $\mathcal{A}$ with binary coproducts such that T^H and T^{H+V} exist. They are called guarded if they factor through a natural transformation $e' : V \to HT^{H+V}$. Uninterpreted solutions are ideal natural transformations $e^\dagger : V \to T^H$ such that $e^\dagger = [\kappa^H, e^\dagger]^\$ \cdot e$ where $[\kappa^H, e^\dagger]^\$: T^{H+V} \to T^H$ is the unique idealized monad morphism extending $[\kappa^H, e^\dagger]$ induced by the freeness property of the completely iterative monad T^{H+V}. We compare these definitions with our Definition 5.1 of NDRPS's:

1. If we "eliminate" the non-determinism from Definition 5.1 by using the identity monad $(\mathrm{Id}, \mathrm{id}, \mathrm{id})$ instead of $(\mathcal{P}^+, \eta^+, \mu^+)$, we obtain a special case of deterministic RPS's as defined in [16] where we restrict to $\mathcal{A} = \mathsf{Set}$ and to finite terms on the right-hand sides of NDRPS's. More precisely, we obtain natural transformations $e : V \to F^{H+V}$ which can be viewed as RPS's $\mathsf{in} \cdot e : V \to T^{H+V}$ where $\mathsf{in} = (\kappa^{H+V})^\# : F^{H+V} \to T^{H+V}$. If e is guarded in the sense of Definition 5.1 (using Id instead of $\mathcal{P}^+$), then $\mathsf{in} \cdot e$ is guarded in the sense of [16].
2. To "eliminate" the non-determinism from our definition of an uninterpreted solution, observe that the identity monad is commutative. By Lemma 3.2 we obtain a canonical distributive law of every polynomial Set-functor H over Id which simply is $\mathrm{id} : H = H\mathrm{Id} \to \mathrm{Id}H = H$, and analogously to Proposition 3.8 this extends to a distributive law of the monads T^H and Id which simply is $\mathrm{id} : T^H = T^H\mathrm{Id} \to \mathrm{Id}T^H = T^H$. The "composite monad" $\mathrm{Id}T^H$ becomes the completely iterative monad T^H, thus the notion of an uninterpreted solution also becomes a special case of the one from [16] (as far as ideal natural transformations $e^\dagger$ are concerned) since $[\kappa^H, e^\dagger]^\$ \cdot \mathsf{in} = [\kappa^H, e^\dagger]^\# : F^{H+V} \to T^H$ by the uniqueness of such monad morphisms extending $\kappa^{H+V} : H + V \to F^{H+V}$.
3. The assumption on uninterpreted solutions of deterministic RPS's to be ideal is necessary to ensure the existence of $[\kappa^H, e^\dagger]^\$$. Working with finite terms in Definition 5.1 has the advantage that we can drop this assumption. In case of a guarded RPS $e : V \to F^{H+V}$ (using Id instead of $\mathcal{P}^+$) uninterpreted solutions automatically are ideal.
4. Technically, the restriction to finite terms on the right-hand sides of NDRPS's is due to the fact that the monad $\mathcal{P}^+ T^H$ is not a completely iterative monad and we thus cannot exploit freeness of the completely iterative monad T^{H+V}. However, since by Theorem 4.3 (and Lemma 4.5) $\mathcal{P}^+ T^H$ is an idealized monad together with a solution operation $(-)^\dagger$ giving canonical (greatest) solutions for guarded equation morphisms, it comes close to a completely iterative monad. In order to capture infinite terms, it would be interesting to see whether $\mathcal{P}^+ T^H$ is something like a "complete Elgot monad" and whether the free completely iterative monad T^{H+V} also is the "free complete Elgot monad". But whereas the concept of Elgot monads has recently been investigated [3], there exist no results for complete Elgot monads.

▶ **Example 5.4.** Consider the NDRPS (1) from the introduction. It is formulated in the classical way using the special binary function symbol or, see e. g. [4], Section II. It can be

viewed as a natural transformation $e : V \to \mathcal{P}^+T^{H+V}$ as follows: according to the signatures of new and given function symbols, we choose the polynomial Set-functors $VX = X$ and $HX = X \times X$. We translate the right-hand term from (1) which is headed by the symbol or into the set containing the two subterms and abstract away from a concrete variable set, obtaining the natural transformation e given by $e_X(\phi(x)) = \{f(x, x), f(x, \phi(x))\}$ for every set X. The naturality states that it is invariant under renaming the variable x.
In classical terms, the NDRPS (1) is a *Greibach scheme* since every new function symbol is part of a term headed by a given function symbol, see e. g. [4], Section IV. Correspondingly, the natural transformation e is guarded since every element of the right-hand set is a term headed by a given operation symbol.
Let us denote the infinite set (2) from the introduction by S. We obtain the natural transformation $e^\dagger$ given by $e^\dagger_X(\phi(x)) = S$ for every set X. Using Remark 5.2, we see that diagram (4) commutes; thus $e^\dagger$ is an uninterpreted solution of e. Similarly, the natural transformation s given by $s_X(\phi(x)) = S \setminus \{t\}$ for every set X is an uninterpreted solution of e, where t is the only infinite tree from S (the rightmost one in (2)).

▶ Remark 5.5. More generally, every classical NDRPS in the sense of Arnold and Nivat [4] can be translated into a NDRPS in the sense of Definition 5.1, using the following ideas:

- the polynomial Set-functors V and H are chosen according to the signatures of new and given function symbols;
- every term headed by the function symbol or is translated to the set of its two subterms;
- given function symbols are distributed over sets using the canonical distributive law $\lambda : H\mathcal{P}^+ \to \mathcal{P}^+H$;
- nested sets are flattened using $\mu^+ : \mathcal{P}^+\mathcal{P}^+ \to \mathcal{P}^+$;
- for every set S occurring in a term headed by a new function symbol an additional new function symbol $\phi_S(x_1, \ldots, x_n)$ with arity according to the number n of variables in S is introduced, S is replaced by $\phi_S(x_1, \ldots, x_n)$ and the equation $\phi_S(x_1, \ldots, x_n) = S$ is added to the NDRPS;
- occurrences of single variables x_i in sets are replaced by $\pi_i(x_1, \ldots, x_n)$ where π_i is an additional given function symbol and the $x_1, \ldots, x_n$ are all variables occurring in the elements of the set (the idea is of course that π_i denotes the i-th projection);
- the natural transformation $e : V \to \mathcal{P}^+F^{H+V}$ constituting the NDRPS is given for every set X and every element from VX by the right-hand side of the equation for the corresponding new function symbol.

In order to obtain a guarded NDRPS from a classical Greibach scheme, it might be necessary to substitute some new function symbols by the right-hand sides of their equations. In conclusion, our notion of a NDRPS covers the classical one, and classical Greibach schemes translate to guarded NDRPS's. Moreover, our notion generalizes the classical one: whereas in [4] NDRPS's define finitely many new operations, and, more important, only allow for finite (nonempty) sets of finite terms on the right-hand sides of NDRPS's, our approach also captures infinitely many newly defined operations and arbitrary (nonempty) sets. It might even be possible to generalize our approach to infinite terms on the right-hand sides, see Remark 5.3(4).

We now state our main result:

▶ **Theorem 5.6.** *Every guarded NDRPS has a canonical greatest uninterpreted solution.*

Before we give the proof of Theorem 5.6, we need to establish an important lemma first. Whenever we write λ or λ' in the rest of the paper, we mean the canonical distributive laws of a polynomial Set-functor H over the monad $\mathcal{P}^+$ from Lemma 3.2 or of T^H over $\mathcal{P}^+$ from

Proposition 3.8. To simplify notation, we denote the free monad F^{H+V} by F for the rest of the paper.

▶ **Definition 5.7.** Given a natural transformation $e' : V \to \mathcal{P}^+HF$, we define the $\bar{\mathcal{H}}$-coalgebra p by

$$p = (\, F \xrightarrow{[\phi^{H+V},\eta^{H+V}]^{-1}} (H+V)F + \mathrm{Id} \xrightarrow{[\eta^+HF\cdot H\eta^{H+V},e']F+\eta^+} \mathcal{P}^+HFF + \mathcal{P}^+ \xrightarrow{\mathcal{P}^+H\mu^{H+V}+\mathcal{P}^+} \mathcal{P}^+HF + \mathcal{P}^+ \xrightarrow{\mathsf{can}} \mathcal{P}^+(HF+\mathrm{Id})\,)\,. \tag{5}$$

By Theorem 4.10 (and Lemma 4.11) there is a (componentwise greatest) natural transformation h such that the diagram

$$\begin{array}{ccc} F & \xrightarrow{h} & T^H \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle J[\tau^H,\eta^H]^{-1}} \\ HF+\mathrm{Id} & \xrightarrow[\bar{\mathcal{H}}h]{} & HT^H+\mathrm{Id} \end{array} \tag{6}$$

commutes (in $[\mathsf{Set},\mathsf{Set}]_{\mathcal{M}}$ for $M = \mathcal{P}^+$). Observe that diagram (6) translates to

$$h = \mathcal{P}^+[\tau^H,\eta^H]\cdot\mu^+(HT^H+\mathrm{Id})\cdot\mathcal{P}^+\mathsf{can}\cdot\mathcal{P}^+(\lambda T^H+\eta^+)\cdot\mathcal{P}^+(Hh+\mathrm{id})\cdot p \tag{7}$$

in $[\mathsf{Set},\mathsf{Set}]$.

▶ **Lemma 5.8.** *The natural transformation $h : F \to \mathcal{P}^+T^H$ from Definition 5.7 is a monad morphism.*

We remark that in the proof of Lemma 5.8, Theorem 4.10 is used to prove the second monad morphism law for h.

Proof of Theorem 5.6 (sketch). The given guarded NDRPS $e : V \to \mathcal{P}^+F$ factors through a natural transformation $e' : V \to \mathcal{P}^+HF$, thus we obtain a natural transformation $h : F \to \mathcal{P}^+T^H$ as in Definition 5.7. We define

$$e^\dagger \equiv (\, V \xrightarrow{\mathsf{inr}} H+V \xrightarrow{\kappa^{H+V}} F \xrightarrow{h} \mathcal{P}^+T^H\,)$$

and prove that this is the componentwise greatest solution of e.

In a first step, one proves that $e^\dagger$ solves e using Lemma 5.8. An important part of this step is to prove that h is the unique monad morphism $[\eta^+T^H\cdot\kappa^H, e^\dagger]^\#$.

In a second step, $e^\dagger$ is proved to be the greatest solution. Here one considers any solution $s : V \to \mathcal{P}^+T^H$ of e. It suffices to show that $x = [\eta^+T^H\cdot\kappa^H, s]^\# : F \to \mathcal{P}^+T^H$ is a coalgebra homomorphism between p and the weakly final $\bar{\mathcal{H}}$-coalgebra over $[\mathsf{Set},\mathsf{Set}]_{\mathcal{M}}$ from Theorem 4.10: since h is known to be the componentwise greatest such homomorphism, it follows $h_X \geq x_X$ for every set X and we conclude

$$e^\dagger_X = \mu^+_{T^HX}\cdot\mathcal{P}^+[\eta^+T^H\cdot\kappa^H, e^\dagger]^\#_X\cdot e_X = \mu^+_{T^HX}\cdot\mathcal{P}^+h_X\cdot e \geq \mu^+_{T^HX}\cdot\mathcal{P}^+x_X\cdot e_X = s_X$$

for every set X using Definition 5.1 and monotonicity of composition in $\mathsf{Set}_{\mathcal{P}^+}$. ◀

▶ **Corollary 5.9.** *For every uninterpreted solution $s : V \to \mathcal{P}^+T^H$ of a NDRPS the sets of all finite cuttings of trees from $s_X(z)$ and $e^\dagger_X(z)$ are the same for every set X and every $z \in VX$.*

Proof. In the second part of the proof of Theorem 5.6 we prove that every solution s of every guarded NDRPS e the monad morphism $[\eta^+ T^H \cdot \kappa^H, s]^\#$ is an $\bar{\mathcal{H}}$-coalgebra homomorphism. According to Lemma 4.11, we have the desired property for this $\bar{\mathcal{H}}$-coalgebra homomorphism and the $\bar{\mathcal{H}}$-coalgebra homomorphism h; this implies that this property also holds for their respective restrictions $s = [\eta^+ T^H \cdot \kappa^H, s]^\# \cdot \kappa^{H+V} \cdot \mathsf{inr}$ and $e^\dagger = h \cdot \kappa^{H+V} \cdot \mathsf{inr}$. ◀

▶ Remark 5.10. The main result of Arnold and Nivat [4] is that greatest solutions of Greibach schemes give the "right" semantics of NDRPS's and can be computed as greatest fixed points. We confirmed the former in Theorem 5.6 and generalized it to a wider class of NDRPS's (cf. Remark 5.5). From our results we also easily recover the latter: restricting to finite sets on the right-hand sides of NDRPS's, the operator $h \mapsto \mathcal{P}^+[\tau^H, \eta^H] \cdot \mu^+(HT^H + \mathrm{Id}) \cdot \mathcal{P}^+\mathsf{can} \cdot \mathcal{P}^+(\lambda T^H + \eta^+) \cdot \mathcal{P}^+(Hh + \mathrm{id}) \cdot p$ on $\mathsf{Set}(F, \mathcal{P}^+ T^H)$ given by equation (7) or equivalently by diagram (6) is componentwise continuous; since we know from Theorem 4.10 and Lemma 4.11 that the greatest fixed point of this operator exists, the second part of Arnold's and Nivat's result follows from (the dual of) Kleene's fixed point theorem. However, the operator is no longer continuous if we allow for infinite sets on the right-hand sides of NDRPS's.

6 Conclusion

We have given a category theoretic definition and semantics of (uninterpreted) NDRPS's. This was achieved by reusing the technical core of Milius and Moss' work on a category theoretic semantics for (ordinary) RPS's [16] and by adding category theoretic concepts that capture the nondeterminism as the nonempty powerset monad and canonical distributive laws over this monad. We showed how our work is related to loc. cit. and that it extends the classical work on NDRPS's by Arnold and Nivat [4].

Although our approach is inspired by [16] and its precursor [9], the non-determinism causes various differences: it is not only more complicated to work with the additional nonempty powerset monad and the canonical distributive laws for it, but many proofs have to be carried out in a more basic setting. For example, the coalgebra functor $\mathcal{H}$ can only be considered on a more basic category, or we even need to use techniques inherent to non-determinism like "determinization" (see the proofs of Proposition 3.5 and Theorem 4.3).

Still, due to the abstract category theoretic framework there are several directions for future generalizations: instead of polynomial functors H it might be possible to use analytic or even weak pullback preserving functors; a starting point is given in Remark 3.3(2). We also suspect that our work can be applied to the environment monad $(-)^E$ instead of $\mathcal{P}^+$ giving an even stronger result (unique solutions) for E-composite RPS's. Technically, our work might be improved by the development of a theory of "complete Elgot monads" as pointed out in Remark 5.3(4). And clearly this paper leaves the question of a category theoretic semantics of interpreted NDRPS's open for future research.

Finally we mention that it is of course possible to admit the empty set in solutions of NDRPS's, i. e. to use the powerset functor $\mathcal{P}$ instead of its nonempty variant $\mathcal{P}^+$. However, this causes a shift in the results since additional least solutions are added: for example it is not difficult to see that every NDRPS where we have recursion in every element of every right-hand set, has a solution where every new function symbol is assigned the empty set. We shall consider this notion of a NDRPS elsewhere.

Acknowledgments The author thanks Stefan Milius and Jiří Adámek for their comments.

References

1 Peter Aczel, Jiří Adámek, Stefan Milius, and Jiří Velebil. Infinite trees and completely iterative theories: A coalgebraic view. *Theoret. Comput. Sci.*, 300:1–45, 2003.

2 Jiří Adámek, Stefan Milius, and Jiří Velebil. Elgot algebras. *Log. Methods Comput. Sci.*, 2(5:4):31 pp., 2006.

3 Jiří Adámek, Stefan Milius, and Jiří Velebil. Elgot theories: a new perspective of the equational properties of iteration. *Math. Structures Comput. Sci.*, 21(2):417–480, 2011.

4 André Arnold and Maurice Nivat. Formal computations of non deterministic recursive program schemes. *Math. Systems Theory*, 13:219–236, 1980.

5 Michael Barr. Coequalizers and free triples. *Math. Z.*, 116:307–322, 1970.

6 Jon Beck. Distributive laws. In *Seminar on Triples and Categorical Homology Theory*, volume 80 of *Lecture Notes in Math.*, pages 119–140. Springer Berlin Heidelberg, 1969.

7 Gérard Boudol. Sémantique opérationelle et algébrique des programmes récursifs non déterministes. These d'Etat, Université de Paris VII, 1980.

8 Bruno Courcelle. Fundamental properties of infinite trees. *Theoret. Comput. Sci.*, 25(2):95–169, 1983.

9 Neil Ghani, Christoph Lüth, and Federico De Marchi. Solving algebraic equations using colagebra. *Theor. Inform. Appl.*, 37:301–314, 2003.

10 Irène Guessarian. *Algebraic Semantics*, volume 99 of *Lecture Notes in Comput. Sci.* Springer, 1981.

11 Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic trace semantics via coinduction. *Log. Methods Comput. Sci.*, 3(4:11):1–36, 2007.

12 Bart Jacobs. Trace semantics for coalgebras. *Electron. Notes Theor. Comput. Sci.*, 106:167–184, 2004.

13 Anders Kock. Monads on symmetric monoidal closed categories. *Arch. Math. (Basel)*, 21:1–10, 1970.

14 Anders Kock. Strong functors and monoidal monads. *Arch. Math. (Basel)*, 23:113–120, 1972.

15 Stefan Milius. Completely iterative algebras and completely iterative monads. *Inform. Comput.*, 196:1–41, 2005.

16 Stefan Milius and Lawrence S. Moss. The category theoretic solution of recursive program schemes. *Theoret. Comput. Sci.*, 366:3–59, 2006.

17 Stefan Milius, Thorsten Palm, and Daniel Schwencke. Complete iterativity for algebras with effects. In *Algebra and Coalgebra in Computer Science (Proc. Third International Conference, Udine, 2009)*, volume 5728 of *Lecture Notes in Comput. Sci.*, pages 34–48. Springer, Berlin, 2009.

18 Philip S. Mulry. Lifting theorems for kleisli categories. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Semantics—9th international conference*, volume 802 of *Lecture Notes in Comput. Sci.*, pages 304–319. Springer Verlag, 1994.

19 Maurice Nivat. On the interpretation of recursive polyadic program schemes. In *Symposia Mathematica XV*, pages 255–281. Academic Press, New York, 1975.

20 Axel Poigné. On effective computations of nondeterministic schemes. In *Symposium on Programming*, volume 137 of *Lecture Notes in Comput. Sci.*, pages 323–336. Springer, 1982.

Step-Indexed Relational Reasoning for Countable Nondeterminism

Jan Schwinghammer[1] and Lars Birkedal[2]

1 **Saarland University**
`jan@ps.uni-saarland.de`
2 **IT University of Copenhagen**
`birkedal@itu.dk`

Abstract

Programming languages with countable nondeterministic choice are computationally interesting since countable nondeterminism arises when modeling fairness for concurrent systems. Because countable choice introduces non-continuous behaviour, it is well-known that developing semantic models for programming languages with countable nondeterminism is challenging. We present a step-indexed logical relations model of a higher-order functional programming language with countable nondeterminism and demonstrate how it can be used to reason about contextually defined may- and must-equivalence. In earlier step-indexed models, the indices have been drawn from ω. Here the step-indexed relations for must-equivalence are indexed over an ordinal greater than ω.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases Countable choice, lambda calculus, program equivalence

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.512

1 Introduction

Programming languages with countable nondeterministic choice are computationally interesting since countable nondeterminism arises when modeling fairness for concurrent systems. In this paper we show how to construct simple semantic models for reasoning about may- and must-equivalence in a call-by-value higher-order functional programming language with countable nondeterminism, recursive types and impredicative polymorphism.

Models for languages with nondeterminism have originally been studied using denotational techniques. In the case of countably branching nondeterminism it is not enough to consider standard ω-continuous complete partial orders and the denotational models become quite involved [3, 6]. This has sparked research in operationally-based theories of equivalence for nondeterministic higher-order languages [1, 10, 11, 12, 13, 18]. In particular, Lassen investigated operationally-based relational methods for countable nondeterminism and suggested that it would be interesting to consider also methods based on logical relations, i.e., where the *types* of the programming languages are given a relational interpretation [10, page 47]. Such an interpretation would allow one to relate terms of different types, as needed for reasoning about parametricity properties of polymorphic types.

For languages with recursive types, however, logical relations cannot be defined by induction on types. In the case of deterministic languages, this problem has been addressed by the technique of syntactic minimal invariance [4] (inspired by domain theory [15]). The idea here is that one proves that a syntactically definable fixed point on a recursive type is contextually equivalent to the identity function, and then uses a so-called unwinding theorem

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 512–524

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

for syntactically definable fixed points when showing the existence of the logical relations. However, in the presence of countable nondeterminism it is not clear how to define the unwindings of the syntactic fixed point in the programming language. Indeed, Lassen proved an unwinding theorem for his language with countable nondeterminism, but he did so by extending the language with new terms needed for representing the unwindings and left open the question of whether this is a conservative extension of the language.

Here we give a logical relations model of our language where we do not rely on syntactic minimal invariance for constructing the logical relations. Instead, we use the idea of step-indexed logical relations [2]. In particular, we show how to use step-indexing over ordinals larger than ω to reason about must-equivalence in the presence of countable nondeterminism.

This approach turns out to be both simple and also useful for reasoning about concrete may- and must-equivalences. We show that our logical relations are sound and complete with respect to the contextually defined notions of may- and must-equivalence. Moreover, we show how to use our logical relations to establish some concrete equivalences. In particular, we prove the recursion-induction rule from Lassen [10] and establish the syntactic minimal invariance property (without extending the language with new unwinding terms). We also include an example to show that the model can be used to prove parametricity properties (free theorems) of polymorphic types.

Overview of the technical development

One way to understand the failure of ω-continuity in an operational setting is to consider the must-convergence predicate $e \Downarrow$, which by Tarski's fixed point theorem can be defined as the least fixed point of the monotone functional $\Phi(R) = \{e \mid \forall e'.\ e \longmapsto e' \ \Rightarrow\ e' \in R\}$ on sets of terms. Here $e \longmapsto e'$ means that e reduces to e' in one step. However, due to the countable branching the fixed point is not reached by ω-many iterations $\bigcup_{n\in\omega} \Phi^n(\emptyset)$. The reason is that even when a program has no infinite reduction sequences, we cannot in general bound the length of reduction sequences by any $n < \omega$.

The idea of step-indexed semantics is a stratified construction of relations which facilitates the interpretation of recursive types, and in previous applications this stratification has typically been realized by indexing over ω. However, as we pointed out, the closure ordinal of the inductively defined must-convergence predicate is strictly larger than ω: the least fixed point $\Downarrow$ is reached after ω_1-many iterations, for ω_1 the least uncountable ordinal. (In fact, the least non-recursive ordinal would suffice [3].) Thus, one of the key steps in our development is the definition of α-indexed uniform relations, for arbitrary ordinals α, in Section 3.

In Section 4 we define a logical ω-indexed uniform relation, and use this relation to prove a CIU theorem for may-contextual equivalence. The logical relation combines step-indexing and biorthogonality, and we can prove that it coincides with may-contextual equivalence; the proofs are similar to those in [17]. Section 5 considers the case of must-contextual equivalence. The only modifications that this requires, compared to Section 4, are the use of ω_1-indexed uniform relations and of a suitably adapted notion of biorthogonality.

In summary, the contribution of this paper is a simple, operationally-based model of countable nondeterminism in a higher-order language, and the use of this model for proving several non-trivial applications in Section 6. In particular, we derive a least-fixed point property for recursive functions in our language, answering a question raised by Lassen [10].

Laird [9] has developed a fully abstract denotational model based on bidomains for a calculus similar to the one studied here but without recursive and polymorphic types; our model appears to be the first model of countable nondeterminism for a language with impredicative polymorphism.

$$\tau ::= \alpha \mid \mathbf{1} \mid \tau_1 \times \tau_2 \mid \tau_1 \to \tau_2 \mid \mu\alpha.\tau_1 + \ldots + \tau_n \mid \forall\alpha.\tau$$
$$v ::= x \mid \langle\rangle \mid \langle v_1, v_2\rangle \mid \lambda x.e \mid \mathsf{in}_i\, v \mid \Lambda\alpha.e$$
$$e ::= v \mid ? \mid \mathsf{proj}_i\, v \mid v\, e \mid \mathsf{case}\ v\ \mathsf{of}\ \mathsf{in}_1\, x_1.\, e_1 \mid \ldots \mid \mathsf{in}_n\, x_n.\, e_n \mid v\, \tau$$
$$E ::= [\,] \mid v\, E$$

Figure 1 Types, terms and evaluation contexts

$$\mathsf{proj}_i\, \langle v_1, v_2\rangle \longmapsto v_i \qquad \mathsf{case}\ (\mathsf{in}_j\, v)\ \mathsf{of}\ (\ldots \mid \mathsf{in}_j\, x_j.\, e_j \mid \ldots) \longmapsto e_j[v/x_j]$$
$$(\lambda x.e)\, v \longmapsto e[v/x] \qquad ? \longmapsto \underline{n} \quad (n \in \mathbb{N})$$
$$(\Lambda\alpha.e)\, \tau \longmapsto e[\tau/\alpha] \qquad v\, e \longmapsto v\, e' \quad \text{if } e \longmapsto e'$$

Figure 2 Operational semantics

2 A lambda calculus with countable choice

Syntax and operational semantics

Figure 1 gives the syntax of a higher-order functional language with recursive and polymorphic types, and a (countably branching) choice construct. We assume disjoint, countably infinite sets of *type variables*, ranged over by α, and *term variables*, ranged over by x. The free type variables of types and terms, $\mathit{ftv}(\tau)$ and $\mathit{ftv}(e)$, and free term variables $\mathit{fv}(e)$, are defined in the usual way. The notation $(\cdot)[\vec{\tau}/\vec{\alpha}]$ denotes the simultaneous capture-avoiding substitution of types $\vec{\tau}$ for the free type variables $\vec{\alpha}$ in types and terms; similarly, $e[\vec{v}/\vec{x}]$ denotes simultaneous capture-avoiding substitution of values $\vec{v}$ for the free term variables $\vec{x}$ in e.

The syntax is kept minimal, and in examples we may use additional syntactic sugar, for instance writing $\mathsf{let}\ x = e\ \mathsf{in}\ e'$ for $(\lambda x.e')\, e$ and $e\, \tau$ for $\mathsf{let}\ f = e\ \mathsf{in}\ f\, \tau$ for some fresh f. We define the unary natural numbers datatype as $\mathsf{nat} = \mu\alpha.\mathbf{1} + \alpha$ and write $\underline{0} = \mathsf{in}_1\, \langle\rangle$ and $\underline{n+1} = \mathsf{in}_2(\underline{n})$. The 'erratic' (finitely branching) choice construct $e_1\ \mathsf{or}\ e_2$ can be defined from $?$ as $\mathsf{let}\ x = ?\ \mathsf{in}\ \mathsf{case}\ x\ \mathsf{of}\ \mathsf{in}_1\, y.\, e_1 \mid \mathsf{in}_2\, y.\, e_2$ for fresh x, y.

The operational semantics of the language is given in Figure 2 by a reduction relation $e \longmapsto e'$. In particular, the choice operator $?$ evaluates nondeterministically to any numeral $\underline{n}$ $(n \in \mathbb{N})$. We also consider evaluation contexts E, and write $E[e]$ for the term obtained by plugging e into E. It is easy to see that $e \longmapsto e'$ holds if and only if $E[e] \longmapsto E[e']$.

Typing judgements take the form $\Delta; \Gamma \vdash e : \tau$ where Γ is a typing context $x_1{:}\tau_1, \ldots, x_n{:}\tau_n$ and where Δ is a finite set of type variables that contains the free type variables of $\tau_1, \ldots, \tau_n$ and τ. The rules defining this judgement are summarized in Figure 3. The typing judgement for evaluation contexts, $\vdash E : \tau \multimap \tau'$, means that $\varnothing; \varnothing \vdash E[e] : \tau'$ holds whenever $\varnothing; \varnothing \vdash e : \tau$.

We write Type for the set of closed types τ, i.e., where $\mathit{ftv}(\tau) = \emptyset$. We write $\mathit{Val}(\tau)$ and $\mathit{Tm}(\tau)$ for the sets of closed values and terms of type τ, resp., and $\mathit{Stk}(\tau)$ for the set of τ-accepting evaluation contexts. For a typing context $\Gamma = x_1{:}\tau_1, \ldots, x_n{:}\tau_n$ with $\tau_1, \ldots, \tau_n \in \mathit{Type}$, let $\mathit{Subst}(\Gamma) = \{\gamma \in \mathit{Val}^{\vec{x}} \mid \forall 1 \le i \le n.\ \gamma(x_i) \in \mathit{Val}(\tau_i)\}$ denote the set of type-respecting value substitutions. In particular, if $\Delta; \Gamma \vdash e : \tau$ then $\varnothing; \varnothing \vdash e\delta\gamma : \tau\delta$ for any $\delta \in \mathit{Type}^{\Delta}$ and $\gamma \in \mathit{Subst}(\Gamma\delta)$, and the type system satisfies the standard progress and preservation theorems.

We let $\mathsf{fix} : \forall\alpha, \beta.((\alpha \to \beta) \to (\alpha \to \beta)) \to (\alpha \to \beta)$ denote a variant of the (call-by-value) fixed point combinator from untyped lambda calculus, $\mathsf{fix} = \Lambda\alpha, \beta.\lambda f.\delta_f(\mathsf{in}\, \delta_f)$ where δ_f

$$\frac{x{:}\tau \in \Gamma \qquad \Delta \vdash \Gamma}{\Delta;\Gamma \vdash x : \tau} \qquad \frac{\Delta \vdash \Gamma}{\Delta;\Gamma \vdash \langle\rangle : \mathbf{1}} \qquad \frac{\Delta;\Gamma \vdash v_1 : \tau_1 \qquad \Delta;\Gamma \vdash v_2 : \tau_2}{\Delta;\Gamma \vdash \langle v_1, v_2\rangle : \tau_1 \times \tau_2}$$

$$\frac{\Delta;\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Delta;\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2} \qquad \frac{\Delta;\Gamma \vdash v : \tau_j[\mu\alpha.\tau_1 + \ldots + \tau_n/\alpha]}{\Delta;\Gamma \vdash \mathsf{in}_j\, v : \mu\alpha.\tau_1 + \ldots + \tau_n}\ 1 \le j \le n$$

$$\frac{\Delta, \alpha;\Gamma \vdash e : \tau}{\Delta;\Gamma \vdash \Lambda\alpha.e : \forall\alpha.\tau} \qquad \frac{\Delta;\Gamma \vdash v : \tau_1 \times \tau_2}{\Delta;\Gamma \vdash \mathsf{proj}_i\, v : \tau_i} \qquad \frac{\Delta;\Gamma \vdash v : \tau' \to \tau \qquad \Delta;\Gamma \vdash e : \tau'}{\Delta;\Gamma \vdash v\, e : \tau}$$

$$\frac{\Delta;\Gamma \vdash v : \mu\alpha.\tau_1 + \ldots + \tau_n \qquad \ldots \qquad \Delta;\Gamma, x_j{:}\tau_j[\mu\alpha.\tau_1 + \ldots + \tau_n/\alpha] \vdash e_j : \tau \qquad \ldots}{\Delta;\Gamma \vdash \mathsf{case}\ v\ \mathsf{of}\ (\ldots \mid \mathsf{in}_j\, x_j.\, e_j \mid \ldots) : \tau}$$

$$\frac{\Delta;\Gamma \vdash v : \forall\alpha.\tau \qquad \Delta \vdash \tau'}{\Delta;\Gamma \vdash v\,\tau' : \tau[\tau'/\alpha]} \qquad \frac{\Delta \vdash \Gamma}{\Delta;\Gamma \vdash ? : \mathsf{nat}}$$

$$\frac{\varnothing \vdash \tau}{\vdash [] : \tau \multimap \tau} \qquad \frac{\varnothing;\varnothing \vdash v : \tau \to \tau_2 \qquad \vdash E : \tau_1 \multimap \tau}{\vdash v\, E : \tau_1 \multimap \tau_2}$$

Figure 3 Typing of terms and evaluation contexts, where $\Gamma ::= \emptyset \mid \Gamma, x{:}\tau$ and $\Delta ::= \emptyset \mid \Delta, \alpha$. The notation $\Delta \vdash \tau$ means that $ftv(\tau) \subseteq \Delta$, and $\Delta \vdash \Gamma$ means that $\Delta \vdash \tau$ holds for all $x{:}\tau \in \Gamma$.

is the term $\lambda y.\mathsf{case}\ y\ \mathsf{of}\ \mathsf{in}\, y'.\, f(\lambda x.\mathsf{let}\ r = y'\, y\ \mathsf{in}\ r\, x)$, and we write $\Omega : \forall\alpha.\alpha$ for the term $\Lambda\alpha.\mathsf{fix}\,\mathbf{1}\,\alpha\,(\lambda f.f)\,\langle\rangle$. Note that reduction from Ω is deterministic and non-terminating.

Contextual approximation

We follow Lassen's approach [10] and define contextual approximation as the largest relation that satisfies certain compatibility and adequacy properties (also see, e.g. [16, 17]). The technical advantage of this approach, compared to the more traditional one of universally quantifying over program contexts, is that in proofs there will be no need to explicitly take care of contexts and of term occurrences within contexts. In our terminology, we keep close to Pitts [16], except for suitably adapting the definitions to take the nondeterministic outcomes of evaluation into account.

The observables on which contextual approximation is based are given by may- and must-convergence. A closed term e *may-converges*, written $e\downarrow$, if $e \longmapsto^* v$ for some $v \in \mathit{Val}$, and e *may-diverges*, written $e\uparrow$, if there is an infinite reduction sequence starting from e. The *must-convergence* predicate $e \Downarrow$ is the complement of may-divergence, and it can be defined inductively by $e \Downarrow$ if and only if for all e', if $e \longmapsto e'$ then $e' \Downarrow$.

▶ **Definition 1** (Type-indexed relation)**.** A *type-indexed relation* is a set of tuples $(\Delta, \Gamma, e, e', \tau)$ such that $\Delta;\Gamma \vdash e : \tau$ and $\Delta;\Gamma \vdash e' : \tau$ holds. We write $\Delta;\Gamma \vdash e\ \mathcal{R}\ e' : \tau$ if $(\Delta, \Gamma, e, e', \tau) \in \mathcal{R}$.

▶ **Definition 2** (Precongruence)**.** A type-indexed relation $\mathcal{R}$ is *reflexive* if $\Delta;\Gamma \vdash e : \tau$ implies $\Delta;\Gamma \vdash e\ \mathcal{R}\ e : \tau$. It is *transitive* if $\Delta;\Gamma \vdash e\ \mathcal{R}\ e' : \tau$ and $\Delta;\Gamma \vdash e'\ \mathcal{R}\ e'' : \tau$ implies $\Delta;\Gamma \vdash e\ \mathcal{R}\ e'' : \tau$. A *precongruence* is a reflexive and transitive type-indexed relation $\mathcal{R}$ that is closed under the inference rules in Figure 4.

▶ **Definition 3** (May- and must-adequate relations)**.** A type-indexed relation $\mathcal{R}$ is *may-adequate* if, whenever $\varnothing;\varnothing \vdash e\ \mathcal{R}\ e' : \tau$ holds, then $e\downarrow$ implies $e'\downarrow$. It is *must-adequate* if, whenever $\varnothing;\varnothing \vdash e\ \mathcal{R}\ e' : \tau$ holds, then $e \Downarrow$ implies $e' \Downarrow$.

$$\frac{}{\Delta;\Gamma \vdash x \mathrel{\mathcal{R}} x : \tau}\ x{:}\tau \in \Gamma \qquad \frac{}{\Delta;\Gamma \vdash \langle\rangle \mathrel{\mathcal{R}} \langle\rangle : \mathbf{1}}$$

$$\frac{\Delta;\Gamma \vdash v_1 \mathrel{\mathcal{R}} v_1' : \tau_1 \qquad \Delta;\Gamma \vdash v_2 \mathrel{\mathcal{R}} v_2' : \tau_2}{\Delta;\Gamma \vdash \langle v_1, v_2\rangle \mathrel{\mathcal{R}} \langle v_1', v_2'\rangle : \tau_1 \times \tau_2} \qquad \frac{\Delta;\Gamma, x{:}\tau_1 \vdash e \mathrel{\mathcal{R}} e' : \tau_2}{\Delta;\Gamma \vdash \lambda x.e \mathrel{\mathcal{R}} \lambda x.e' : \tau_1 \to \tau_2}$$

$$\frac{\Delta;\Gamma \vdash v \mathrel{\mathcal{R}} v' : \tau_j[\mu\alpha.\tau_1 + \ldots + \tau_n/\alpha]}{\Delta;\Gamma \vdash \mathsf{in}_j\, v \mathrel{\mathcal{R}} \mathsf{in}_j\, v' : \mu\alpha.\tau_1 + \ldots + \tau_n}\ 1 \le j \le n \qquad \frac{\Delta,\alpha;\Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau}{\Delta;\Gamma \vdash \Lambda\alpha.e \mathrel{\mathcal{R}} \Lambda\alpha.e' : \forall\alpha.\tau}$$

$$\frac{\Delta;\Gamma \vdash v \mathrel{\mathcal{R}} v' : \tau_1 \times \tau_2}{\Delta;\Gamma \vdash \mathsf{proj}_i\, v \mathrel{\mathcal{R}} \mathsf{proj}_i\, v' : \tau_i} \qquad \frac{\Delta;\Gamma \vdash v \mathrel{\mathcal{R}} v' : \tau' \to \tau \qquad \Delta;\Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau'}{\Delta;\Gamma \vdash v\, e \mathrel{\mathcal{R}} v'\, e' : \tau}$$

$$\frac{\Delta;\Gamma \vdash v \mathrel{\mathcal{R}} v' : \tau \quad \ldots \quad \Delta;\Gamma, x_j{:}\tau_j[\tau/\alpha] \vdash e_j \mathrel{\mathcal{R}} e_j' : \tau' \quad \ldots}{\Delta;\Gamma \vdash \mathsf{case}\ v\ \mathsf{of}\ (\ldots \mid \mathsf{in}_j\, x_j.\, e_j \mid \ldots) \mathrel{\mathcal{R}} \mathsf{case}\ v'\ \mathsf{of}\ (\ldots \mid \mathsf{in}_j\, x_j.\, e_j \mid \ldots) : \tau'}\ \tau = \mu\alpha.\tau_1 + \ldots + \tau_n$$

$$\frac{\Delta;\Gamma \vdash v \mathrel{\mathcal{R}} v' : \forall\alpha.\tau}{\Delta;\Gamma \vdash v\,\tau' \mathrel{\mathcal{R}} v'\,\tau' : \tau[\tau'/\alpha]}\ ftv(\tau') \subseteq \Delta \qquad \frac{}{\Delta;\Gamma \vdash ? \mathrel{\mathcal{R}} ? : \mathsf{nat}}$$

Figure 4 Compatibility properties of type-indexed relations

▶ **Definition 4** (Contextual approximations and equivalences). *May-contextual approximation*, written $\lesssim_{\downarrow}^{ctx}$, is the largest may-adequate precongruence. *May-contextual equivalence*, $\cong_{\downarrow}^{ctx}$, is the symmetrization of $\lesssim_{\downarrow}^{ctx}$. Analogously, *must-contextual approximation*, written $\lesssim_{\Downarrow}^{ctx}$, is the largest must-adequate precongruence, and *must-contextual equivalence*, $\cong_{\Downarrow}^{ctx}$, is its symmetrization. *Contextual approximation*, $\lesssim^{ctx}$, and *contextual equivalence*, $\cong^{ctx}$, are given as intersections of the respective may- and must-relations, and thus $\cong^{ctx}$ is also the symmetrization of $\lesssim^{ctx}$.

That this largest (may-, must-) adequate precongruence exists can be shown as in [16], by proving that the relation $S = \bigcup\{R \mid R \text{ compatible and (may-, must-) adequate}\}$ is an adequate precongruence.

In principle, to establish an equivalence $\Delta;\Gamma \vdash e \cong^{ctx} e' : \tau$ it suffices to find some may- and must-adequate congruence $\mathcal{R}$ that contains the tuple $(\Delta, \Gamma, e, e', \tau)$ since $\cong^{ctx}$ is the largest such relation. However, in practice it is difficult to verify that a relation $\mathcal{R}$ has the necessary compatibility properties in Figure 4. An alternative characterization of the contextual approximation and equivalence relations can be given in terms of CIU preorders [14], which we define next.

▶ **Definition 5** (CIU preorders). *May- and must-CIU preorder*, written $\lesssim_{\downarrow}^{ciu}$ and $\lesssim_{\Downarrow}^{ciu}$ resp., are the type-indexed relations defined as follows: for all e, e' with $\Delta;\Gamma \vdash e : \tau$ and $\Delta;\Gamma \vdash e' : \tau$,

- $\Delta;\Gamma \vdash e \lesssim_{\downarrow}^{ciu} e' : \tau \;\Leftrightarrow\; \forall\delta \in \mathit{Type}^{\Delta}, \gamma \in \mathit{Subst}(\Gamma\delta), E \in \mathit{Stk}(\tau\delta).\ E[e\delta\gamma]\downarrow \Rightarrow E[e'\delta\gamma]\downarrow$
- $\Delta;\Gamma \vdash e \lesssim_{\Downarrow}^{ciu} e' : \tau \;\Leftrightarrow\; \forall\delta \in \mathit{Type}^{\Delta}, \gamma \in \mathit{Subst}(\Gamma\delta), E \in \mathit{Stk}(\tau\delta).\ E[e\delta\gamma]\Downarrow \Rightarrow E[e'\delta\gamma]\Downarrow$

The CIU preorder is defined as the intersection of $\lesssim_{\downarrow}^{ciu}$ and $\lesssim_{\Downarrow}^{ciu}$.

▶ **Theorem 6** (CIU theorem). *The (may-, must-) CIU preorder coincides with (may-, must-) contextual approximation.*

Using the CIU theorem, it is easy to verify that all the deterministic reductions are also valid equivalences, and that the various call-by-value eta laws hold. Moreover, we can

$$\mathsf{let}\ x = ?\ \mathsf{in}\ e \cong^{ctx} e \quad (x \notin fv(e)) \qquad \mathsf{let}\ x = v\ \mathsf{in}\ e \cong^{ctx} e[v/x] \qquad \mathsf{let}\ x = e\ \mathsf{in}\ x \cong^{ctx} e$$

$$e\ \mathsf{or}\ e \cong^{ctx} e \qquad \Omega \lesssim^{ctx}_{\downarrow} e \qquad \Omega \lesssim^{ctx}_{\Downarrow} e$$

$$e_1\ \mathsf{or}\ e_2 \cong^{ctx} e_2\ \mathsf{or}\ e_1 \qquad e_1 \lesssim^{ctx}_{\downarrow} e_1\ \mathsf{or}\ e_2 \qquad e_1\ \mathsf{or}\ e_2 \lesssim^{ctx}_{\Downarrow} e_1$$

$$(e_1\ \mathsf{or}\ e_2)\ \mathsf{or}\ e_3 \cong^{ctx} e_1\ \mathsf{or}\ (e_2\ \mathsf{or}\ e_3) \qquad e\ \mathsf{or}\ \Omega \cong^{ctx}_{\downarrow} e \qquad e\ \mathsf{or}\ \Omega \cong^{ctx}_{\Downarrow} \Omega$$

Figure 5 Basic may- and must-theory, for $e_1\ \mathsf{or}\ e_2 \equiv \mathsf{let}\ x = ?\ \mathsf{in}\ \mathsf{case}\ x\ \mathsf{of}\ \mathsf{in}_1\, y.\, e_1 \mid \mathsf{in}_2\, y.\, e_2$

establish the laws of Moggi's computational lambda calculus and the basic (inequational) theory of erratic choice (Figure 5). We will prove the CIU theorem in Section 4 (for the may-CIU preorder) and Section 5 (for the must-CIU preorder).

3 Uniform relations

For an ordinal number α and a set X we define an *α-indexed uniform relation on X* to be a family $(R_\beta)_{\beta<\alpha}$ of relations $R_\beta \subseteq X$ such that

- $R_0 = X$,
- $R_{\beta+1} \subseteq R_\beta$ for all $\beta < \alpha$, and
- $R_\lambda = \bigcap_{\beta<\lambda} R_\beta$ for every limit ordinal $\lambda < \alpha$.

Let $Rel_\alpha(X)$ denote the α-indexed uniform relations on X.

Recursive definitions

The notions of n-equivalence, non-expansiveness and contractiveness (e.g., [5]) all generalize from the case of ω-indexed uniform relations: Given α-indexed uniform relations $R, S \in Rel_\alpha(X)$ and $\nu < \alpha$ we say that R and S are *ν-equivalent*, written $R \overset{\nu}{=} S$, if $R_\beta = S_\beta$ for all $\beta \leq \nu$. In particular, $R = S$ if and only if $R \overset{\nu}{=} S$ for all $\nu < \alpha$.

A function $F : Rel_\alpha(X_1) \times \cdots \times Rel_\alpha(X_n) \to Rel_\alpha(X)$ is *non-expansive* if $\vec{R} \overset{\nu}{=} \vec{S}$ implies $F(\vec{R}) \overset{\nu}{=} F(\vec{S})$, and F is *contractive* if $\vec{R} \overset{\nu}{=} \vec{S}$ implies $F(\vec{R}) \overset{\nu+1}{=} F(\vec{S})$. If $R \in Rel_\alpha(X)$ then $\triangleright R \in Rel_\alpha(X)$ is the uniform relation determined by $\triangleright R_{\beta+1} = R_\beta$; this operation gives rise to a contractive function on $Rel_\alpha(X)$.

▶ **Proposition 7** (Unique fixed points). *If $F : Rel_\alpha(X) \to Rel_\alpha(X)$ is contractive, then F has a unique fixed point $fix r.F(r)$.*

Proof. First note that F has at most one fixed point: if R, S are fixed points of F then, by the contractiveness of F, we can establish that $R = F(R) \overset{\nu}{=} F(S) = S$ holds for all $\nu < \alpha$ by induction and thus $R = S$.

Because of the uniformity conditions it is sufficient to give the components of the fixed point $fix r.F(r)$ that are indexed by successor ordinals. We set $fix r.F(r)_{\nu+1} = F(R)_{\nu+1}$ where $R \in Rel_\alpha(X)$ is defined by $R_\beta = fix r.F(r)_\beta$ for $\beta \leq \nu$ and $R_\beta = \emptyset$ for $\beta > \nu$. By induction, it is easy to see that $fix r.F(r) \in Rel_\alpha(X)$ and that $F(fix r.F(r))_\nu = fix r.F(r)_\nu$ holds for all $\nu < \alpha$, and thus $F(fix r.F(r)) = fix r.F(r)$. ◀

Proposition 7 is an instance of Di Gianantonio and Miculan's sheaf-theoretic fixed point theorem [7]. Indeed, an α-indexed uniform relation on X corresponds to a subobject of the constant sheaf on X in the sheaf topos on α.

Uniform relations on syntax

For $\tau, \tau' \in \mathit{Type}$ we consider the collections of α-indexed uniform relations between values, terms and evaluation contexts: we write $\mathit{VRel}_\alpha(\tau, \tau')$ for $\mathit{Rel}_\alpha(\mathit{Val}(\tau) \times \mathit{Val}(\tau'))$, $\mathit{SRel}_\alpha(\tau, \tau')$ for $\mathit{Rel}_\alpha(\mathit{Stk}(\tau) \times \mathit{Stk}(\tau'))$, and $\mathit{TRel}_\alpha(\tau, \tau')$ for $\mathit{Rel}_\alpha(\mathit{Tm}(\tau) \times \mathit{Tm}(\tau'))$.

The description of the logical relations in the sections below makes use of the following (non-expansive) constructions on uniform relations:

- $R_1 \times R_2 \in \mathit{VRel}_\alpha(\tau_1 \times \tau_2, \tau_1' \times \tau_2')$, for $R_1 \in \mathit{VRel}_\alpha(\tau_1, \tau_1')$ and $R_2 \in \mathit{VRel}_\alpha(\tau_2, \tau_2')$, is defined by $(R_1 \times R_2)_\beta = \{(\langle v_1, v_2\rangle, \langle v_1', v_2'\rangle) \mid (v_1, v_1') \in (R_1)_\beta \ \wedge \ (v_2, v_2') \in (R_2)_\beta\}$.
- $R_1 \to R_2 \in \mathit{VRel}_\alpha(\tau_1 \to \tau_2, \tau_1' \to \tau_2')$, for $R_1 \in \mathit{VRel}_\alpha(\tau_1, \tau_1')$ and $R_2 \in \mathit{TRel}_\alpha(\tau_2, \tau_2')$, is given by $(R_1 \to R_2)_\beta = \{(\lambda x.e, \lambda x.e') \mid \forall \nu \le \beta.\, \forall (v, v') \in (R_1)_\nu.\, (e[v/x], e'[v'/x]) \in (R_2)_\nu\}$.
- $\forall r.F(r) \in \mathit{VRel}_\alpha(\forall\alpha.\tau_1, \forall\alpha.\tau_1')$, for $F_{\tau,\tau'} : \mathit{VRel}_\alpha(\tau, \tau') \to \mathit{TRel}_\alpha(\tau_1[\tau/\alpha], \tau_1'[\tau'/\alpha])$ a family of non-expansive maps, is the uniform relation that is defined by $\forall r.F(r)_\beta = \{(\Lambda\alpha.e, \Lambda\alpha.e') \mid \forall \tau, \tau' \in \mathit{Type}, R \in \mathit{VRel}_\alpha(\tau, \tau').\ (e[\tau/\alpha], e'[\tau'/\alpha]) \in F_{\tau,\tau'}(R)_\beta\}$.
- $\mathsf{in}_j R \in \mathit{VRel}_\alpha(\tau, \tau')$, for $\tau = \mu\alpha.\tau_1 + \ldots + \tau_m$ and $\tau' = \mu\alpha.\tau_1' + \ldots + \tau_n'$ and $R \in \mathit{VRel}_\alpha(\tau_j[\tau/\alpha], \tau_j'[\tau'/\alpha])$, is given by $(\mathsf{in}_j R)_\beta = \{(\mathsf{in}_j\, v, \mathsf{in}_j\, v') \mid (v, v') \in R_\beta\}$.

4 May equational theory

In this section, we will define a logical uniform relation that is used to prove that may-CIU preorder and may-contextual approximation coincide. The key idea of the definition is the usual one of step-indexing [2], i.e., that the observables can be stratified based on step-counting in the operational semantics. We write $e \downarrow_n$ if $e \longmapsto \ldots \longmapsto v$ for some $v \in \mathit{Val}$ in at most n reduction steps, thus $e \downarrow$ holds if and only if $e \downarrow_n$ for some n.

Logical ω-indexed uniform relation for may-approximation

In the case of may-approximation, it suffices to consider ω-indexed uniform relations. Using the constructions on relations given above, we define a relational interpretation $[\![\tau]\!]\,(\vec{r}) \in \mathit{VRel}_\omega(\tau[\vec{\tau}/\vec{\alpha}], \tau[\vec{\tau}'/\vec{\alpha}])$ by induction on the type $\vec{\alpha} \vdash \tau$, given closed types $\tau_1, \tau_1', \ldots, \tau_k, \tau_k' \in \mathit{Type}$ and relations $r_1 \in \mathit{VRel}_\omega(\tau_1, \tau_1'), \ldots, r_k \in \mathit{VRel}_\omega(\tau_k, \tau_k')$:

$$[\![\alpha_i]\!]\,(\vec{r}) = r_i \qquad [\![\tau_1 \times \tau_2]\!]\,(\vec{r}) = [\![\tau_1]\!]\,(\vec{r}) \times [\![\tau_2]\!]\,(\vec{r})$$

$$[\![\mathbf{1}]\!]\,(\vec{r}) = (\mathit{Id}_{\mathbf{1}})_{n<\omega} \qquad [\![\tau_1 \to \tau_2]\!]\,(\vec{r}) = [\![\tau_1]\!]\,(\vec{r}) \to [\![\tau_2]\!]\,(\vec{r})^{\perp\!\!\perp}$$

$$[\![\forall\alpha.\tau]\!]\,(\vec{r}) = \forall r.[\![\tau]\!]\,(\vec{r}, r)^{\perp\!\!\perp} \qquad [\![\mu\alpha.\tau_1 + \ldots + \tau_m]\!]\,(\vec{r}) = \mathit{fix}\, s.\, \bigcup_j \mathsf{in}_j(\triangleright\, [\![\tau_j]\!]\,(\vec{r}, s))$$

Here, value relations $r \in \mathit{VRel}_\omega(\tau, \tau')$ are lifted to relations $r^\perp \in \mathit{SRel}_\omega(\tau, \tau')$ on evaluation contexts and to relations $r^{\perp\!\!\perp} \in \mathit{TRel}_\omega(\tau, \tau')$ on terms by biorthogonality, much as in [8]:

$$r_n^\perp = \{(E, E') \mid \forall j \le n.\, \forall (v, v') \in r_j.\ E[v] \downarrow_j \Rightarrow\ E'[v'] \downarrow\}$$
$$r_n^{\perp\!\!\perp} = \{(e, e') \mid \forall j \le n.\, \forall (E, E') \in r_j^\perp.\ E[e] \downarrow_j \Rightarrow\ E'[e'] \downarrow\}$$

The fixed point in the interpretation of recursive types is well-defined by Proposition 7 since each $[\![\tau]\!]$ denotes a family of non-expansive functions, and thus composition with $\triangleright$ yields a contractive function.

The following observation is useful for calculations:

▶ **Lemma 8** (Context composition). *If* $(v, v') \in [\![\tau_1 \to \tau_2]\!]\,\vec{r}_n$ *and* $(E, E') \in [\![\tau_2]\!]\,\vec{r}_n^{\perp}$ *then* $(E[v\,[]], E'[v'\,[]]) \in [\![\tau_1]\!]\,\vec{r}_{n+1}^{\perp}$.

Proof. Let $j \leq n+1$, $(v_1, v_1') \in [\![\tau_1]\!]\,\vec{r}_j$. Assume $E[v\,v_1]\downarrow_j$. We have $v = \lambda x.e$ and $v' = \lambda x.e'$ and $(\lambda x.e, \lambda x.e') \in [\![\tau_1 \to \tau_2]\!]\,\vec{r}_n$ for some x, e, e' and necessarily $E[v\,v_1] \longmapsto E[e[v_1/x]]\downarrow_{j-1}$. By definition, $(e[v_1/x], e'[v_1'/x]) \in [\![\tau_2]\!]\,\vec{r}_{j-1}^{\perp\perp}$. From $(E, E') \in [\![\tau_2]\!]\,\vec{r}_n^{\perp}$ we obtain $E'[e'[v_1'/x]]\downarrow$. Thus, $E'[v'\,v_1']\downarrow$. ◀

The relational interpretation extends pointwise to value substitutions: $(\gamma, \gamma') \in [\![\Gamma]\!]\,\vec{r}_n$ if $(\gamma(x), \gamma(x')) \in [\![\tau]\!]\,\vec{r}_n$ for all $x{:}\tau \in \Gamma$. Based on this interpretation we consider the following type-indexed relation:

$$\Delta;\Gamma \vdash e \lesssim_{\downarrow}^{log} e' : \tau \quad \text{where } \Delta = \vec{\alpha}$$
$$\Leftrightarrow\ \forall \vec{\tau}, \vec{\tau}'.\, \forall \vec{r} \in VRel_\omega(\vec{\tau}, \vec{\tau}').\, \forall n < \omega.\, \forall (\gamma, \gamma') \in [\![\Gamma]\!]\,\vec{r}_n.\ (e[\vec{\tau}/\vec{\alpha}]\gamma, e'[\vec{\tau}'/\vec{\alpha}]\gamma') \in [\![\tau]\!]\,\vec{r}_n^{\perp\perp}$$

The definition of $\lesssim_{\downarrow}^{log}$ builds in enough closure properties to prove its compatibility.

▶ **Proposition 9** (Fundamental property). *The relation $\lesssim_{\downarrow}^{log}$ has the compatibility properties given in Figure 4. In particular, it is reflexive: if $\Delta;\Gamma \vdash e : \tau$ then $\Delta;\Gamma \vdash e \lesssim_{\downarrow}^{log} e : \tau$.*

Proof. We consider the inference rules from Figure 4 in turn.

- For the introduction of recursive types, we assume $\Delta;\Gamma \vdash v \lesssim_{\downarrow}^{log} v' : \tau_j[\mu\alpha.\tau_1 + \ldots + \tau_m/\alpha]$ and $1 \leq j \leq m$, and then prove that $\Delta;\Gamma \vdash \mathsf{in}_j\, v \lesssim_{\downarrow}^{log} \mathsf{in}_j\, v' : \mu\alpha.\tau_1 + \ldots + \tau_m$.
  For notational convenience we only consider the case of closed terms. Let τ abbreviate the type $\mu\alpha.\tau_1 + \ldots + \tau_m$. Note that $[\![\tau]\!]\,\vec{r} = \bigcup_j \mathsf{in}_j\big(\triangleright\,[\![\tau_j]\!]\,(\vec{r}, [\![\tau]\!]\,\vec{r})\big) = \bigcup_j \mathsf{in}_j\big(\triangleright\,[\![\tau_j[\tau/\alpha]]\!]\,(\vec{r})\big)$ by definition and a substitution lemma, and that $[\![\tau_j[\tau/\alpha]]\!]\,(\vec{r}) \subseteq \triangleright\,[\![\tau_j[\tau/\alpha]]\!]\,(\vec{r})$. Thus, assuming $(E, E') \in [\![\tau]\!]\,\vec{r}_n^{\perp}$ it follows from Lemma 8 that $(E[(\lambda x.\mathsf{in}_j\, x)\,[]], E'[(\lambda x.\mathsf{in}_j\, x)\,[]]) \in [\![\tau_j[\tau/\alpha]]\!]\,\vec{r}_{n+1}^{\perp}$. Thus, if $E[\mathsf{in}_j\, v]\downarrow_i$ for some $i \leq n$ then $E'[(\lambda x.\mathsf{in}_j\, x)\,v'])\downarrow$ follows from $(v, v') \in [\![\tau_j[\tau/\alpha]]\!]\,\vec{r}_{n+1}^{\perp\perp}$. Therefore we can conclude $E'[\mathsf{in}_j\, v']\downarrow$, and we have shown $(\mathsf{in}_j\, v, \mathsf{in}_j\, v') \in [\![\tau]\!]\,\vec{r}_n^{\perp\perp}$. Since n was chosen arbitrarily, we have $\Delta;\Gamma \vdash \mathsf{in}_j\, v \lesssim_{\downarrow}^{log} \mathsf{in}_j\, v' : \tau$.
- For the elimination of recursive types, we assume that τ is of the form $\mu\alpha.\tau_1 + \ldots + \tau_m$, $\Delta;\Gamma, x_j{:}\tau_j[\tau/\alpha] \vdash e_j \lesssim_{\downarrow}^{log} e_j' : \tau'$ for all $1 \leq j \leq m$ and $\Delta;\Gamma \vdash v \lesssim_{\downarrow}^{log} v' : \tau$. We prove $\Delta;\Gamma \vdash \mathsf{case}\ v\ \mathsf{of}(\ldots|\mathsf{in}_j\, x_j.\, e_j|\ldots) \lesssim_{\downarrow}^{log} \mathsf{case}\ v'\ \mathsf{of}(\ldots|\mathsf{in}_j\, x_j.\, e_j'|\ldots) : \tau'$.
  For simplicity we only consider the case of closed terms. By definition and by a substitution lemma we have $[\![\tau]\!]\,\vec{r} = \bigcup_j \mathsf{in}_j\big(\triangleright\,[\![\tau_j]\!]\,(\vec{r}, [\![\tau]\!]\,\vec{r})\big) = \bigcup_j \mathsf{in}_j(\triangleright\,[\![\tau_j[\tau/\alpha]]\!]\,\vec{r})$. Moreover, $(\lambda x.\mathsf{case}\ x\ \mathsf{of}(\ldots|\mathsf{in}_j\, x_j.\, e_j|\ldots), \lambda x.\mathsf{case}\ x\ \mathsf{of}(\ldots|\mathsf{in}_j\, x_j.\, e_j'|\ldots)) \in [\![\tau \to \tau']\!]\,\vec{r}_n$ for any n. To see this, assume $k \leq n$, let $(a, a') \in [\![\tau]\!]\,\vec{r}_n$ and $(E, E') \in [\![\tau']\!]\,\vec{r}_n^{\perp}$ such that $E[\mathsf{case}\ a\ \mathsf{of}(\ldots|\mathsf{in}_j\, x_j.\, e_j|\ldots)]\downarrow_k$. By the above observation we have $a = \mathsf{in}_j a_j$ and $a' = \mathsf{in}_j a_j'$ for some $(a_j, a_j') \in [\![\tau_j[\tau/\alpha]]\!]\,\vec{r}_{k-1}$. From $E[\mathsf{case}\ a\ \mathsf{of}(\ldots|\mathsf{in}_j\, x_j.\, e_j|\ldots)]\downarrow_k$ we obtain $E[e_j[a_j/x_j]]\downarrow_{k-1}$, and thus the assumption on e_j, e_j' gives $E'[e_j'[a_j'/x_j]]\downarrow$ from which we can conclude $E'[\mathsf{case}\ a'\ \mathsf{of}(\ldots|\mathsf{in}_j\, x_j.\, e_j'|\ldots)]\downarrow$.
  To prove the case, assume next that $(E, E') \in [\![\tau']\!]\,\vec{r}_n^{\perp}$. From Lemma 8 we obtain $(E[(\lambda x.\mathsf{case}\ x\ \mathsf{of}(\ldots|\mathsf{in}_j\, x_j.\, e_j|\ldots))\,[]], E'[(\lambda x.\mathsf{case}\ x\ \mathsf{of}(\ldots|\mathsf{in}_j\, x_j.\, e_j'|\ldots))\,[]]) \in [\![\tau]\!]\,\vec{r}_{n+1}^{\perp}$. Since $(v, v') \in [\![\tau]\!]\,\vec{r}_{n+1}^{\perp\perp}$ by assumption, we obtain that $E[\mathsf{case}\ v\ \mathsf{of}(\ldots|\mathsf{in}_j\, x_j.\, e_j|\ldots)]\downarrow_n$ implies $E[\mathsf{case}\ v'\ \mathsf{of}(\ldots|\mathsf{in}_j\, x_j.\, e_j'|\ldots)]\downarrow$ as required.
- For choice, we assume $\Delta \vdash \Gamma$ and show $\Delta;\Gamma \vdash ? \lesssim_{\downarrow}^{log} ? : \mathsf{nat}$. Suppose $(E, E') \in [\![\mathsf{nat}]\!]\,\vec{r}_n^{\perp}$ and $E[?]\downarrow_j$ for some $j \leq n$. Then $E[?] \longmapsto E[\underline{k}]$ and $E[\underline{k}]\downarrow_{j-1}$ for some $k \in \mathbb{N}$. By induction on k, and using the compatibility for the introduction of recursive types, we obtain that $(\underline{k}, \underline{k}) \in [\![\mathsf{nat}]\!]\,\vec{r}_n^{\perp\perp}$, and thus $E'[\underline{k}]\downarrow$. Hence $E'[?]\downarrow$.

The proofs for the remaining rules are similar. ◀

▶ **Theorem 10** (Coincidence). *$\Delta;\Gamma \vdash e \lesssim_{\downarrow}^{log} e' : \tau$ if and only if $\Delta;\Gamma \vdash e \lesssim_{\downarrow}^{ciu} e' : \tau$.*

$$\frac{\Delta;\Gamma \vdash v \mathrel{\mathcal{R}} v' : \tau \qquad \Delta;\Gamma, x{:}\tau \vdash e \mathrel{\mathcal{R}} e' : \tau'}{\Delta;\Gamma \vdash e[v/x] \mathrel{\mathcal{R}} e'[v'/x] : \tau'} \qquad \frac{\Delta,\alpha;\Gamma \vdash e \mathrel{\mathcal{R}} e' : \tau'}{\Delta;\Gamma[\tau/\alpha] \vdash e \mathrel{\mathcal{R}} e' : \tau'[\tau/\alpha]}\ \Delta \vdash \tau$$

Figure 6 Substitutivity properties of type-indexed relations

Proof. For the direction from left to right, let $\delta \in \mathit{Type}^{\Delta}$, $\gamma \in \mathit{Subst}(\Gamma\delta)$ and $E \in \mathit{Stk}(\tau\delta)$, and assume $E[e\delta\gamma]\downarrow$, i.e., $E[e\delta\gamma]\downarrow_n$ for some n. We must show $E[e'\delta\gamma]\downarrow$. As a consequence of Proposition 9, $(\gamma,\gamma) \in [\![\Gamma\delta]\!]_n$ and $(E,E) \in [\![\tau\delta]\!]_n^{\perp}$. By definition of $\Delta;\Gamma \vdash e \lesssim_{\downarrow}^{log} e' : \tau$ and a substitution lemma we have $(e\delta\gamma, e'\delta\gamma) \in [\![\tau\delta]\!]_n^{\perp\perp}$, and thus $E[e\delta\gamma]\downarrow_n$ gives $E[e'\delta\gamma]\downarrow$.

For the direction from right to left, first note that the logical relation is closed under may-CIU approximation; more precisely, if $\Delta;\Gamma \vdash e \lesssim_{\downarrow}^{log} e' : \tau$ and $\Delta;\Gamma \vdash e' \lesssim_{\downarrow}^{ciu} e'' : \tau$ then $\Delta;\Gamma \vdash e \lesssim_{\downarrow}^{log} e'' : \tau$. This observation follows from the definition of $(\cdot)^{\perp\perp}$ used in $\Delta;\Gamma \vdash e \lesssim_{\downarrow}^{log} e' : \tau$ and the definition of CIU approximation. Now assume that $\Delta;\Gamma \vdash e \lesssim_{\downarrow}^{ciu} e' : \tau$. By Proposition 9, $\Delta;\Gamma \vdash e \lesssim_{\downarrow}^{log} e : \tau$, and thus $\Delta;\Gamma \vdash e \lesssim_{\downarrow}^{log} e' : \tau$. ◀

Proof of CIU Theorem 6(1). We first show that $\lesssim_{\downarrow}^{ciu}$ is contained in $\lesssim_{\downarrow}^{ctx}$. By definition, $\lesssim_{\downarrow}^{ctx}$ is the largest may-adequate precongruence, thus it is sufficient to establish that $\lesssim_{\downarrow}^{ciu}$ is a may-adequate precongruence. From the definition it is immediate that $\lesssim_{\downarrow}^{ciu}$ is may-adequate, reflexive and transitive. By Theorem 10, $\lesssim_{\downarrow}^{ciu}$ coincides with $\lesssim_{\downarrow}^{log}$ which is compatible by Proposition 9.

For the other direction, following Pitts [17], we first consider the special case where $\varnothing;\varnothing \vdash e \lesssim_{\downarrow}^{ctx} e' : \tau$. To prove $\varnothing;\varnothing \vdash e \lesssim_{\downarrow}^{ciu} e' : \tau$, note that $\varnothing;\varnothing \vdash E[e] \lesssim_{\downarrow}^{ctx} E[e'] : \tau'$ holds for all evaluation contexts E such that $\vdash E : \tau \multimap \tau'$ since $\lesssim_{\downarrow}^{ctx}$ is reflexive and compatible. Hence, that $E[e]\downarrow$ implies $E[e']\downarrow$ follows since $\lesssim_{\downarrow}^{ctx}$ is may-adequate.

The general case reduces to this special case since may-contextual approximation has the substitutivity properties given in Figure 6. For the first of these, assume $\Delta;\Gamma \vdash v \lesssim_{\downarrow}^{ctx} v' : \tau$ and $\Delta;\Gamma, x{:}\tau \vdash e \lesssim_{\downarrow}^{ctx} e' : \tau'$. From the definition of may-CIU approximation it is easy to see

$$\Delta;\Gamma \vdash e[v/x] \lesssim_{\downarrow}^{ciu} (\lambda x.e)\, v : \tau' \quad \text{and} \quad \Delta;\Gamma \vdash (\lambda x.e')\, v' \lesssim_{\downarrow}^{ciu} e'[v'/x] : \tau' \ .$$

Since we have already shown that $\lesssim_{\downarrow}^{ciu}$ is contained in $\lesssim_{\downarrow}^{ctx}$, and since $\Delta;\Gamma \vdash (\lambda x.e)\, v \lesssim_{\downarrow}^{ctx} (\lambda x.e')\, v' : \tau'$ by compatibility, we can conclude $\Delta;\Gamma \vdash e[v/x] \lesssim_{\downarrow}^{ctx} e'[v'/x] : \tau'$ by transitivity. The second substitutivity property is proved similarly, using a weakening property of may-contextual approximation. ◀

5 Must equational theory

To define the logical relation for must-approximation, we need to stratify the observables again. For terms e and ordinals β we define $e \Downarrow_\beta$ inductively, as the least relation such that $e \Downarrow_\beta$ if for all e' such that $e \longmapsto e'$ there exists $\nu < \beta$ and $e' \Downarrow_\nu$. The essential observation is that $\Downarrow_\beta$ indeed captures must-convergent behaviour.

▶ **Proposition 11** (Stratified must-convergence). *$e \Downarrow$ if and only if $e \Downarrow_\beta$ for some $\beta < \omega_1$ (for ω_1 the least uncountable ordinal).*

Proof. The proof from left to right is by induction on $e \Downarrow$. By induction hypothesis there exists ordinals $\nu(e') < \omega_1$ for each term e' such that $e \longmapsto e'$. Let $\beta = \bigcup \nu(e')$, then

$\beta+1<\omega_1$ (since there are only countably many such e' and each $\nu(e')$ is countable) and $e \Downarrow_{\beta+1}$. The direction from right to left is by induction on β. ◀

Logical ω_1-indexed uniform relation for must-approximation

Proposition 11 indicates that logical relations for must-approximation need to be indexed over ω_1. The lifting of value relations $r \in \mathit{VRel}_{\omega_1}(\tau,\tau')$ to relations $r^{\perp} \in \mathit{SRel}_{\omega_1}(\tau,\tau')$ on evaluation contexts and to relations $r^{\perp\perp} \in \mathit{TRel}_{\omega_1}(\tau,\tau')$ on terms is defined with respect to must termination.

$$r^{\perp}_{\beta} = \{(E,E') \mid \forall \nu \leq \beta.\ \forall (v,v') \in r_\nu.\ E[v] \Downarrow_\nu \Rightarrow\ E'[v'] \Downarrow\ \}$$
$$r^{\perp\perp}_{\beta} = \{(e,e') \mid \forall \nu \leq \beta.\ \forall (E,E') \in r^{\perp}_\nu.\ E[e] \Downarrow_\nu \Rightarrow\ E'[e'] \Downarrow\ \}$$

Except for this difference, the relational interpretation $[\![\tau]\!]\,(\vec{r}) \in \mathit{VRel}_{\omega_1}(\tau[\vec{\tau}/\vec{\alpha}], \tau[\vec{\tau}'/\vec{\alpha}])$ is literally the same as in Section 4 and defined by induction on the type $\vec{\alpha} \vdash \tau$, given closed types $\tau_1,\tau_1',\ldots,\tau_k,\tau_k' \in \mathit{Type}$ and relations $r_1 \in \mathit{VRel}_{\omega_1}(\tau_1,\tau_1'),\ldots,r_k \in \mathit{VRel}_{\omega_1}(\tau_k,\tau_k')$:

$$[\![\alpha_i]\!]\,(\vec{r}) = r_i \qquad [\![\tau_1 \times \tau_2]\!]\,(\vec{r}) = [\![\tau_1]\!]\,(\vec{r}) \times [\![\tau_2]\!]\,(\vec{r})$$
$$[\![\mathbf{1}]\!]\,(\vec{r}) = (\mathit{Id}_{\mathbf{1}})_{\beta<\omega_1} \qquad [\![\tau_1 \to \tau_2]\!]\,(\vec{r}) = [\![\tau_1]\!]\,(\vec{r}) \to [\![\tau_2]\!]\,(\vec{r})^{\perp\perp}$$
$$[\![\forall\alpha.\tau]\!]\,(\vec{r}) = \forall r.[\![\tau]\!]\,(\vec{r},r)^{\perp\perp} \qquad [\![\mu\alpha.\tau_1+\ldots+\tau_m]\!]\,(\vec{r}) = \mathit{fix}\, s.\textstyle\bigcup_j \mathsf{in}_j(\triangleright\,[\![\tau_j]\!]\,(\vec{r},s))$$

Logical must-approximation is defined as follows:

$$\Delta;\Gamma \vdash e \lesssim^{\mathit{log}}_{\Downarrow} e' : \tau \ \text{ where } \Delta = \vec{\alpha}$$
$$\Leftrightarrow\ \forall \vec{\tau},\vec{\tau}'.\forall \vec{r} \in \mathit{VRel}_{\omega_1}(\vec{\tau},\vec{\tau}').\forall \beta<\omega_1.\forall(\gamma,\gamma') \in [\![\Gamma]\!]\,\vec{r}_\beta.\ (e[\vec{\tau}/\vec{\alpha}]\gamma, e'[\vec{\tau}'/\vec{\alpha}]\gamma') \in [\![\tau]\!]\,\vec{r}^{\perp\perp}_{\beta}$$

▶ **Proposition 12** (Fundamental property). *The relation $\lesssim^{\mathit{log}}_{\Downarrow}$ has the compatibility properties given in Figure 4. In particular, it is reflexive: if $\Delta;\Gamma \vdash e : \tau$ then $\Delta;\Gamma \vdash e \lesssim^{\mathit{log}}_{\Downarrow} e : \tau$.*

Proof. The proof is similar to the one for Proposition 9. We give only the case for choice, where we assume $\Delta \vdash \Gamma$ and prove $\Delta;\Gamma \vdash\ ?\ \lesssim^{\mathit{log}}_{\Downarrow}\ ?\ :\ \mathsf{nat}$. Suppose $(E,E') \in [\![\mathsf{nat}]\!]\,\vec{r}^{\perp}_{\beta}$ and $E[?] \Downarrow_\beta$. Then $E[?] \longmapsto e$ implies that e is of the form $E[\underline{k}]$ and $E[\underline{k}] \Downarrow_{\nu_k}$ for some $k \in \mathbb{N}$ and $\nu_k < \beta$. Using the compatibility for the introduction form of recursive types, an induction on k shows that $(\underline{k},\underline{k}) \in [\![\mathsf{nat}]\!]\,\vec{r}^{\perp\perp}_{\nu_k}$, and thus $E'[\underline{k}] \Downarrow$ for all $k \in \mathbb{N}$. Hence $E'[?] \Downarrow$. ◀

▶ **Theorem 13** (Coincidence). *$\Delta;\Gamma \vdash e \lesssim^{\mathit{log}}_{\Downarrow} e' : \tau$ if and only if $\Delta;\Gamma \vdash e \lesssim^{\mathit{ciu}}_{\Downarrow} e' : \tau$.*

Proof. The proof is completely analogous to that of Theorem 10. For the direction from left to right one uses the characterization of $\Downarrow$ in terms of $\Downarrow_\beta$ (Proposition 11) and then appeals to Proposition 12. The direction from right to left uses the fact that $\lesssim^{\mathit{log}}_{\Downarrow}$ is closed under must-CIU approximation. ◀

Proof of CIU Theorem 6(2). The proof is analogous to that of Theorem 6(1). From the definition, $\lesssim^{\mathit{ciu}}_{\Downarrow}$ is a must-adequate reflexive and transitive relation, by Proposition 12 and Theorem 13 it is also compatible, and thus contained in $\lesssim^{\mathit{ctx}}_{\Downarrow}$. From this containment and the closure of $\lesssim^{\mathit{ciu}}_{\Downarrow}$ under beta conversion it follows that $\lesssim^{\mathit{ctx}}_{\Downarrow}$ has the substitutivity properties in Figure 6. Thus it suffices to prove the containment of $\lesssim^{\mathit{ctx}}_{\Downarrow}$ in $\lesssim^{\mathit{ciu}}_{\Downarrow}$ for closed terms, which is clear by the compatibility and must-adequacy of $\lesssim^{\mathit{ctx}}_{\Downarrow}$. ◀

$$\frac{\Delta;\Gamma \vdash v\, v' \lesssim^{ctx}_{\downarrow} v' : \tau_1 \to \tau_2}{\Delta;\Gamma \vdash \mathsf{fix}\,\tau_1\tau_2\, v \lesssim^{ctx}_{\downarrow} v' : \tau_1 \to \tau_2} \qquad \frac{\Delta;\Gamma \vdash v\, v' \lesssim^{ctx}_{\Downarrow} v' : \tau_1 \to \tau_2}{\Delta;\Gamma \vdash \mathsf{fix}\,\tau_1\tau_2\, v \lesssim^{ctx}_{\Downarrow} v' : \tau_1 \to \tau_2}$$

Figure 7 Recursion induction: least fixed point property of fix

6 Applications

This section illustrates how the logical relation characterization of contextual approximation can be used to derive interesting examples and further proof principles. We consider three such applications: a recursion-induction principle for recursively defined functions, syntactic minimal invariance of a recursive type, and a "free theorem" about a polymorphic type.

Proving recursion-induction for a similar language (without polymorphic types) has been an open problem [10]. Here, the proof is essentially a straightforward induction, using the indexing of the logical relations.

Recursion-induction

Recall from the introduction that $\mathsf{fix} : \forall\alpha,\beta.((\alpha\to\beta)\to(\alpha\to\beta))\to(\alpha\to\beta)$ is given by the term $\Lambda\alpha,\beta.\lambda f.\delta_f(\mathsf{in}\,\delta_f)$ where δ_f is the term $\lambda y.\mathsf{case}\ y\ \mathsf{of}\ \mathsf{in}\,y'.\, f(\lambda x.(\lambda r.r\,x)(y'\,y))$. We now prove that fix is a *least* fixed point combinator, i.e., we prove the soundness of the recursion-induction rules in Figure 7. We only include the proof for $\lesssim^{ctx}_{\Downarrow}$ and for notational simplicity we assume that the contexts Δ and Γ are empty. We assume the premise of the rule, and to show the conclusion we first prove that $(h,v') \in [\![\tau_1\to\tau_2]\!]_\beta$ where h is $\lambda x.(\lambda r.r\,x)\,(\delta_v\,(\mathsf{in}\,\delta_v))$, for all $\beta < \omega_1$. The result then follows from the agreement of the logical relation with contextual approximation and transitivity, since $\mathsf{fix}\,\tau_1\tau_2\,v \cong^{ctx} v\,h \lesssim^{ctx}_{\Downarrow} v\,v' \lesssim^{ctx}_{\Downarrow} v'$.

To prove $(h,v') \in [\![\tau_1\to\tau_2]\!]_\beta$ we proceed by induction on β and assume that $(h,v') \in [\![\tau_1\to\tau_2]\!]_{\nu_1}$, for all $\nu<\beta$; we are then to show that $(h,v') \in [\![\tau_1\to\tau_2]\!]_\beta$. From the typing rules, v' must be of the form $\lambda x.e'$ for some e'. So let $\beta_1 \le \beta$ and $(u,u') \in [\![\tau_1]\!]_{\beta_1}$, then it remains to show $((\lambda r.r\,u)(\delta_v\,(\mathsf{in}\,\delta_v)), e'[u'/x]) \in [\![\tau_2]\!]^{\perp\!\perp}_{\beta_1}$.

Suppose $\beta_2 \le \beta_1$, $(E,E') \in [\![\tau_2]\!]^{\perp}_{\beta_2}$ and $E[(\lambda r.r\,u)(\delta_v\,(\mathsf{in}\,\delta_v))] \Downarrow_{\beta_2}$; we are to show $E'[e'[u'/x]] \Downarrow$. By (the must-analogue of) Lemma 8 and the fundamental property of the logical relation applied to v we obtain $(E[(\lambda r.r\,u)\,((\lambda x.v\,x)\,[])], E'[(\lambda r.r\,u')\,((\lambda x.v\,x)\,[])]) \in [\![\tau_1\to\tau_2]\!]^{\perp}_{\beta_2}$. Then, since $\delta_v\,(\mathsf{in}\,\delta_v) \longmapsto^2 v\,h$ and $(\lambda x.v\,x)\,h \longmapsto v\,h$, we have $E[(\lambda r.r\,u)(v\,h)] \Downarrow_{\beta_3}$ for $\beta_3 < \beta_2 \le \beta$, and hence also $E'[(\lambda r.r\,u')\,(v\,v')] \Downarrow$ by induction hypothesis.

By the premise and Theorem 13 we have that $v\,v'$ CIU-approximates v', and thus we get $E'[(\lambda r.r\,u')\,v'] \Downarrow$. Finally, since $(\lambda r.r\,u')\,v' \longmapsto^* e'[u'/x]$ we obtain the required $E'[e'[u'/x]] \Downarrow$.

Syntactic minimal invariance

Consider the type $\tau = \mu\alpha.\mathsf{nat} + \alpha \to \alpha$. Let $id = \lambda x.x$ and consider the term

$$f \;\equiv\; \lambda h,x.\mathsf{case}\ x\ \mathsf{of}\ \mathsf{in}_1\,y.\,\mathsf{in}_1\,y \mid \mathsf{in}_2\,g.\,\mathsf{in}_2\,\lambda y.h(g(h\,y))\ .$$

We shall show that $\mathsf{fix}\,\tau\tau\,f \cong^{ctx} id : \tau\to\tau$. This equivalence corresponds to the characterization of solutions to recursive domain equations as minimal invariants in domain-theoretic work [15], from which Pitts derives several (co-) induction principles.

By the soundness of the call-by-value beta- and eta-laws for contextual equivalence (Figure 5) and the transitivity of $\lesssim^{ctx}$, it is easy to see that $f\,id \cong^{ctx} id : \tau\to\tau$. The recursion-induction principle therefore yields $\mathsf{fix}\,\tau\tau\,f \lesssim^{ctx} id : \tau\to\tau$.

For the reverse approximation we first show $id \lesssim^{log}_{\Downarrow} h : \tau\to\tau$ where h is again the term $\lambda x.(\lambda r.r\,x)(\delta_f\,(\mathsf{in}\,\delta_f))$. We show this by proving $(id,h) \in [\![\tau\to\tau]\!]_\beta$ for all $\beta<\omega_1$ by induction

on β. (The case for may-approximation is similar.) By definition, we need to show that for all $\nu \leq \beta$ and all $(v, v') \in [\![\tau]\!]_\nu$, $(id\, v, h\, v') \in [\![\tau]\!]_\nu^{\perp\!\!\perp}$. Since $[\![\tau]\!] = \mathsf{in}_1(\triangleright [\![\mathsf{nat}]\!]) \cup \mathsf{in}_2(\triangleright [\![\tau \to \tau]\!])$ there are two cases to consider:

- Case $(v, v') \in \mathsf{in}_1(\triangleright [\![\mathsf{nat}]\!])_\nu$. Then there exist $u, u' \in \mathit{Val}(\mathsf{nat})$ such that $v = \mathsf{in}_1\, u$, $v' = \mathsf{in}_1\, u'$ and $(u, u') \in [\![\mathsf{nat}]\!]_{\nu'}$ for all $\nu' < \nu \leq \beta$. Note that $(\lambda x.(\lambda r.r\, x)(\delta_f\, (\mathsf{in}\, \delta_f)))\, v' \cong^{ctx} v' : \tau$ in this case. Thus, given $(E, E') \in [\![\tau]\!]_\nu^\perp$ such that $E[id\, v] \Downarrow_\nu$, it suffices to show $E'[v'] \Downarrow$ which easily follows from $(v, v') \in [\![\tau]\!]_\nu$.
- Case $(v, v') \in \mathsf{in}_2(\triangleright [\![\tau \to \tau]\!])_\nu$. Then there exist $g, g' \in \mathit{Val}(\tau \to \tau)$ such that $v = \mathsf{in}_2\, g$, $v' = \mathsf{in}_2\, g'$ and $(g, g') \in [\![\tau \to \tau]\!]_{\nu'}$ for all $\nu' < \nu \leq \beta$. In this case, we have the equivalence $(\lambda x.(\lambda r.r\, x)(\delta_f\, (\mathsf{in}\, \delta_f)))\, v' \cong^{ctx} \mathsf{in}_2(\lambda y.h(g'(h\, y))) : \tau$. Thus, it suffices to show $(g, \lambda y.h(g'(h\, y))) \in [\![\tau \to \tau]\!]_{\nu'}$ for all $\nu' < \nu$, or equivalently, $(g\, u, h(g'(h\, u'))) \in [\![\tau]\!]_{\nu'}^{\perp\!\!\perp}$ for all $\nu' < \nu$ and all $(u, u') \in [\![\tau]\!]_{\nu'}$. Let $(E, E') \in [\![\tau]\!]_{\nu'}^\perp$ and suppose $E[g\, u] \Downarrow_{\nu'}$; we have to show $E'[h(g'(h\, u'))] \Downarrow$. From the induction hypothesis we obtain $(E[id\, []], E'[h\, []]) \in [\![\tau]\!]_{\nu'+1}^\perp$, and thus $(E, E'[h\, []]) \in [\![\tau]\!]_{\nu'}^\perp$. Since $(g, g') \in [\![\tau \to \tau]\!]_{\nu'}$ the latter entails $(E[g\, []], E'[h(g'\, [])]) \in [\![\tau]\!]_{\nu'}^\perp$. Now, applying the induction hypothesis again this shows $(E[g(id\, [])], E'[h(g'(h\, []))]) \in [\![\tau]\!]_{\nu'+1}^\perp$, and thus the assumptions $E[g\, u] \Downarrow_{\nu'}$ and $(u, u') \in [\![\tau]\!]_{\nu'}$ imply $E'[h(g'(h\, u))] \Downarrow$.

By Theorem 13 and the CIU theorem, $id \lesssim_\Downarrow^{log} h : \tau \to \tau$ implies $id \lesssim_\Downarrow^{ctx} h : \tau \to \tau$. Since $id \cong^{ctx} f\, id : \tau \to \tau$ and $f\, h \cong^{ctx} \mathsf{fix}\, \tau\tau\, f : \tau \to \tau$ we obtain $id \lesssim_\Downarrow^{ctx} \mathsf{fix}\, \tau\tau\, f : \tau \to \tau$ by compatibility and transitivity of must-contextual equivalence.

Parametricity

Let $\tau_1, \tau_2 \in \mathit{Type}$ be closed types. Then the contextual approximation

$$\varnothing; h{:}\forall\alpha.\alpha \times \alpha \to \alpha, f{:}\tau_1 \to \tau_2, x{:}\tau_1, y{:}\tau_1 \;\vdash\; h\, \tau_2\, \langle f\, x, f\, y\rangle \lesssim^{ctx} f(h\, \tau_1\, \langle x, y\rangle) : \tau_2\ . \tag{1}$$

holds. For the proof of (1), we will consider the case of must-approximation only (may-approximation is completely analogous) and show

$$\varnothing; h{:}\forall\alpha.\alpha \times \alpha \to \alpha, f{:}\tau_1 \to \tau_2, x{:}\tau, y{:}\tau \;\vdash\; h\, \tau_2\, \langle f\, x, f\, y\rangle \lesssim_\Downarrow^{log} f(h\, \tau_1\, \langle x, y\rangle) : \tau_2\ .$$

Fix $\beta < \omega_1$, $h \in \mathit{Val}(\forall\alpha.\alpha \times \alpha \to \alpha)$, $f \in \mathit{Val}(\tau_1 \to \tau_2)$ and $x, y \in \mathit{Val}(\tau_1)$. We need to show

$$(h\, \tau_2\, \langle f\, x, f\, y\rangle, f(h\, \tau_1\, \langle x, y\rangle)) \in [\![\tau_2]\!]_\beta^{\perp\!\!\perp}\ . \tag{2}$$

We have $(h, h) \in [\![\forall\alpha.\alpha \times \alpha \to \alpha]\!]_\beta^{\perp\!\!\perp}$ by Proposition 12, and we will instantiate α by (the opposite of) the graph of f. More precisely, consider the relation $r \in \mathit{VRel}(\tau_2, \tau_1)$ given by $r_\nu = \{(v, v') \mid (v, f\, v') \in [\![\tau_2]\!]_{\nu+1}^{\perp\!\!\perp}\}$. Note that we have $(id, f) \in [\![\alpha \to \tau_2]\!]\, r_\beta$. Hence, to prove (2) it suffices to show $(h\, \tau_2\, \langle f\, x, f\, y\rangle, h\, \tau_1\, \langle x, y\rangle) \in r_\beta^{\perp\!\!\perp}$.

By definition of the logical relation we have $(h\, \tau_2, h\, \tau_1) \in [\![\alpha \times \alpha \to \alpha]\!]\, r_\beta^{\perp\!\!\perp}$, and by the compatibility properties it remains to show $(f\, x, x) \in r_\beta^{\perp\!\!\perp}$ and $(f\, y, y) \in r_\beta^{\perp\!\!\perp}$. We consider the former: Let $(E, E') \in r_\nu^\perp$ for $\nu \leq \beta$ such that $E[f\, x] \Downarrow_\nu$; we must prove $E'[x] \Downarrow$. We have $(f, id) \in [\![\tau_1 \to \alpha]\!]\, r_\nu$ from which $(E[f\, []], E'[]) \in [\![\tau_1]\!]_\nu^\perp$ follows. By Proposition 12 we have $(x, x) \in [\![\tau_1]\!]_\nu^{\perp\!\!\perp}$, and thus $E[f\, x] \Downarrow_\nu$ implies $E'[x] \Downarrow$.

Let us now consider the reverse approximation of (1), which holds under the condition that f is total and deterministic, i.e., that for all $v \in \mathit{Val}(\tau_1)$ there exists $u \in \mathit{Val}(\tau_2)$ such that $f\, v \cong^{ctx} u : \tau_2$.

We proceed as above and show only for the case of must-approximation. For $\beta < \omega_1$, $h \in \mathit{Val}(\forall\alpha.\alpha \times \alpha \to \alpha)$, $f \in \mathit{Val}(\tau_1 \to \tau_2)$ and $x, y \in \mathit{Val}(\tau_1)$ we will prove

$$(f(h\, \tau_1\, \langle x, y\rangle), h\, \tau_2\, \langle f\, x, f\, y\rangle) \in [\![\tau_2]\!]_\beta^{\perp\!\!\perp}\ . \tag{3}$$

We use $(h,h) \in [\![\forall\alpha.\alpha\times\alpha\to\alpha]\!]^{\perp\!\!\perp}_{\beta}$ where we instantiate α by the relation $s \in \mathit{VRel}(\tau_1,\tau_2)$, given by $s_\nu = \{(v,v') \mid (f\,v,v') \in [\![\tau_2]\!]^{\perp\!\!\perp}_{\nu+1}\}$. First note that $(f,id) \in [\![\alpha\to\tau_2]\!]\,s_\beta$, and thus the proof of (3) reduces to showing $(h\,\tau_1\,\langle x,y\rangle, h\,\tau_2\,\langle f\,x, f\,y\rangle) \in s^{\perp\!\!\perp}_{\beta}$.

Since we have $(h\,\tau_1, h\,\tau_2) \in [\![\alpha\times\alpha\to\alpha]\!]\,s^{\perp\!\!\perp}_{\beta}$ it suffices to show $(x, f\,x) \in s^{\perp\!\!\perp}_{\beta}$ and $(y, f\,y) \in s^{\perp\!\!\perp}_{\beta}$, and we consider the former. Let $(E,E') \in s^{\perp}_{\nu}$ for $\nu \le \beta$ such that $E[x]\Downarrow_\nu$; we must prove $E'[f\,x]\Downarrow$. By the assumption that f is total there exists $u \in \mathit{Val}(\tau_2)$ such that $f\,x \cong^{ctx} u : \tau_2$, and so it suffices to prove $E'[u]\Downarrow$. But this follows from $(x,u) \in s_\nu$, and the latter is immediate from the definition of s.

References

1 Gul Agha, Ian A. Mason, Scott F. Smith, and Carolyn L. Talcott. A foundation for actor computation. *J. Funct. Program.*, 7(1):1–72, 1997.

2 Andrew W. Appel and David A. McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.*, 23(5):657–683, 2001.

3 Krzysztof R. Apt and Gordon D. Plotkin. Countable nondeterminism and random assignment. *J. ACM*, 33(4):724–767, 1986.

4 Lars Birkedal and Robert Harper. Relational interpretations of recursive types in an operational setting. *Inf. Comput.*, 155(1-2):3–63, 1999.

5 Lars Birkedal, Bernhard Reus, Jan Schwinghammer, Kristian Støvring, Jacob Thamsborg, and Hongseok Yang. Step-indexed Kripke models over recursive worlds. In *POPL*, pages 119–132, 2011.

6 Pietro Di Gianantonio, Furio Honsell, and Gordon D. Plotkin. Uncountable limits and the lambda calculus. *Nord. J. Comput.*, 2(2):126–145, 1995.

7 Pietro Di Gianantonio and Marino Miculan. Unifying recursive and co-recursive definitions in sheaf categories. In *FOSSACS*, pages 136–150, 2004.

8 Derek Dreyer, Georg Neis, and Lars Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *ICFP*, pages 143–156, 2010.

9 James Laird. Bidomains and full abstraction for countable nondeterminism. In *FOSSACS*, pages 352–366, 2006.

10 Søren B. Lassen. *Relational Reasoning about Functions and Nondeterminism.* PhD thesis, University of Aarhus, 1998.

11 Søren B. Lassen and Andrew Moran. Unique fixed point induction for McCarthy's amb. In *MFCS*, pages 198–208, 1999.

12 Søren B. Lassen and Corin Pitcher. Similarity and bisimilarity for countable nondeterminism and higher-order functions. *Electr. Notes Theor. Comput. Sci.*, 10, 1997.

13 Paul Blain Levy. Infinitary Howe's method. In *CMCS*, pages 85–104, 2006.

14 Ian A. Mason and Carolyn L. Talcott. Equivalence in functional languages with effects. *J. Funct. Program.*, 1(3):287–327, 1991.

15 Andrew M. Pitts. Relational properties of domains. *Inf. Comput.*, 127(2):66–90, 1996.

16 Andrew M. Pitts. Typed operational reasoning. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 7, pages 245–289. MIT Press, 2005.

17 Andrew M. Pitts. Step-indexed biorthogonality: a tutorial example. In *Modelling, Controlling and Reasoning About State*, Dagstuhl Seminar Proceedings, 2010.

18 David Sabel and Manfred Schmidt-Schauß. A call-by-need lambda calculus with locally bottom-avoiding choice: context lemma and correctness of transformations. *Math. Struct. Comp. Sci.*, 18(3):501–553, 2008.

Algebraic Characterization of the Alternation Hierarchy in $FO^2[<]$ on Finite Words*

Howard Straubing

Computer Science Department, Boston College
Chestnut Hill, Massachusetts, USA 02467
straubin@cs.bc.edu

Abstract

We give an algebraic characterization of the quantifier alternation hierarchy in first-order two-variable logic on finite words. As a result, we obtain a new proof that this hierarchy is strict. We also show that the first two levels of the hierarchy have decidable membership problems, and conjecture an algebraic decision procedure for the other levels.

1998 ACM Subject Classification F.4.1 Mathematical Logic; F.4.3 Formal Languages

Keywords and phrases Automata, finite model theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.525

1 Introduction

We study first-order sentences interpreted in finite words over a finite alphabet Σ, with the single relation $<$ on positions in the word. It is well known (Kamp [6], Immerman and Kozen [5]) that every such sentence is equivalent to one in which only three variables are used. There has been extensive study, from the standpoint of first-order and temporal logic, automata theory, and algebra, of the fragment $FO^2[<]$ of sentences that use only two variables. (See, for example, Ettesami, Vardi and Wilke [4]; Schwentick, Thérien and Vollmer [13]; Straubing and Thérien [16]. Tesson and Thérien [17] give a broad-ranging survey of the many places in which the class of languages definable in this logic arises.)

Weis and Immerman [20] examined the hierarchy within $FO^2[<]$ based on alternation of quantifiers. Using model-theoretic methods, they showed that this hierarchy is strict. Kufleitner and Weil [9] show that each level of the hierarchy defines a variety of languages. This implies, among other things, that whether a regular language $L \subseteq \Sigma^*$ can be defined by a sentence of a given level k in the hierarchy is completely determined by the syntactic monoid $M(L)$ of L. While they do not provide an explicit algebraic description of the levels, Kufleitner and Weil do show that one can effectively compute the alternation depth of a given language in $FO^2[<]$ with an error no more than 1.

Here we give an exact algebraic characterization of each level of the alternation hierarchy; that is, we give an algebraic description of sequence $\mathbf{V}_n$ of families of finite monoids with the property that L is defined by a sentence with k quantifier alternations if and only if $M(L) \in \mathbf{V}_k$. Our characterization is in terms of the two-sided semidirect product of finite monoids and of pseudovarieties of finite monoids. More precisely, we show that the k^{th} level of the hierarchy corresponds to the weakly iterated two-sided semidirect product of k copies of the pseudovariety $\mathbf{J}$ of $\mathcal{J}$-trivial monoids. While many algebraic decompositions of the pseudovariety $\mathbf{DA}$ corresponding to $FO^2[<]$ have been studied, and while it has always been

* Research partially supported by National Science Foundation Grant CCF-0915065

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 525–537

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

clear that **DA** is equal to the closure of **J** under two-sided semidirect product, the fact that the levels of this hierarchy have such a simple logical significance appears to be new.

This still leaves the question of whether the membership problem for each of the levels is decidable. This is precisely the kind of question that algebraic methods are best suited to answer, since it is often possible to reduce the problem to one of verifying identities in the syntactic monoid. We produce a sequence of identities, based on work of Almeida and Weil [2], that we conjecture characterizes membership in each of the levels of the hierarchy. We show that these identities are necessary conditions, and use this fact to give a new proof of the strictness of the alternation hierarchy. The identities are known to be sufficient for the first two levels, which gives algebraic decision procedures for determining whether a give regular language is definable by a two-variable sentence with one or two quantifier alternations.

We present general preliminaries in Section 2 and particulars about two-sided semidirect products in Section 3. We prove our characterization theorem in Section 4; the argument is an adaptation of one given in [16]. We apply the result to questions of strictness and decidability in Section 5.

2 Logical And Algebraic Preliminaries

We review these preliminaries briefly and somewhat informally. The books by Pin [10] and Straubing [15] are references for all the matters discussed here.

2.1 First-order logic

Let Σ be a finite alphabeuntitled foldert. We build first-order formulas from atomic formulas $x < y$ and $Q_\sigma x$, where $\sigma \in \Sigma$. These formulas are interpreted in finite words over Σ: variables are interpreted as positions, with $x < y$ indicating that position x is strictly to the left of position y, and $Q_\sigma x$ indicating that the letter in position x is σ. A *sentence* (a formula without free variables) accordingly *defines* the *language* $L \subseteq \Sigma^*$ of all words w that satisfy the sentence. For example, if $\Sigma = \{\sigma, \tau\}$, then the set of words in which both σ and τ occur, and in which there are at least two occurrences of σ to the left of the first occurrence of τ is defined by the sentence

$$\exists x(Q_\tau x \wedge \forall y(y < x \to Q_\sigma y) \wedge \exists z_1 \exists z_2(z_1 < z_2 \wedge z_2 < x)).$$

As mentioned in the introduction, every first-order sentence of this kind is equivalent to one in which there are only three variables, provided we are allowed to reuse variable symbols. Here we are concerned with the languages definable by sentences of the logic we denote by $FO^2[<]$, consisting of formulas in which only two variables are used. This logic is known to have strictly weaker expressive power than full first-order logic. Note however, that the language in the example above is definable in this restricted logic, by the sentence

$$\exists x(Q_\tau x \wedge \forall y(y < x \to Q_\sigma y) \wedge \exists y(y < x \wedge \exists x(x < y))).$$

We cannot use standard constructions to write such sentences in prefix form without increasing the number of variables. Nonetheless, it is still possible to describe a different sort of normal form that will allow us to define the depth of quantifier alternation in a formula. We allow atomic formulas with $\leq$ as well as $<$, replace every occurrence of $\neg Q_\sigma x$ by

$$\bigvee_{\tau \in \Sigma \setminus \{\sigma\}} Q_\tau x,$$

and apply DeMorgan's Laws to move negations past conjunctions, disjunctions and quantifiers. We obtain as a result an equivalent formula that contains only existential and universal quantifiers, and the connectives $\wedge$ and $\vee$, with no occurrences of negation. This does not change the number of disuntitled foldertinct variable symbols. Consider the parse tree of a formula in this form. Every interior node is labeled by either $\wedge$, $\vee$, or a quantifier. Consider just the sequence of quantifiers on a path from the root to a leaf: this sequence contains alternating blocks of existential and universal quantifiers. The maximum number of such blocks over all paths in the tree is the *alternation depth* of the formula. For example, the sentence displayed above has alternation depth 2. We write $FO^2_n[<]$ for the fragment of $FO^2[<]$ consisting of formulas with alternation depth no more than n.

2.2 Finite monoids

A *monoid* is a set M together with an associative operation for which there is an identity element $1 \in M$. If Σ is an alphabet, then Σ^* is a monoid with concatenation of words as the multiplication. Σ^* is the *free monoid* on Σ: this means that every map $\alpha : \Sigma \to M$, where M is a monoid, extends in a unique fashion to a homomorphism from Σ^* into M.

Apart from free monoids, all the monoids we consider in this paper are finite. If M is a finite monoid, and $m \in M$, then there is a unique $e \in \{m^k : k > 1\}$ that is *idempotent, i.e.,* $e^2 = e$. We denote this element m^ω.

If M, N are monoids then we say M *divides* N, and write $M \prec N$, if M is a homomorphic image of a submonoid of N.

We are interested in monoids because of their connection with automata and regular languages: A *congruence* on Σ^* is an equivalence relation $\sim$ on Σ^* such that $u_1 \sim u_2$, $v_1 \sim v_2$, implies $u_1v_1 \sim u_2v_2$. The classes of $\sim$ then form a monoid $M = \Sigma^*/\sim$, and the map $u \mapsto [u]_\sigma$ sending each word to its congruence class is a homomorphism from Σ^* onto M. If $L \subseteq \Sigma^*$, then $\equiv_L$, the *syntactic congruence* of L, is the coarsest congruence for which L is a union of congruence classes. The quotient monoid $\Sigma^*/\equiv_L$ is called the *syntactic monoid* of L and is denoted $M(L)$.

We say that a monoid M *recognizes* a language $L \subseteq \Sigma^*$ if there is a homomorphism $\alpha : \Sigma^* \to M$ and a subset X of M such that $\alpha^{-1}(X) = L$. The following proposition gives the fundamental properties linking automata to finite monoids.

▶ **Proposition 1.**
Let $L \subseteq \Sigma^*$.

- *A monoid M recognizes L if and only if $M(L) \prec M$.*
- *L is a regular language if and only if $M(L)$ is finite.*

2.3 Varieties and identities

A collection **V** of finite monoids closed under finite direct products and division is called a *pseudovariety* of finite monoids. (The prefix 'pseudo' is because of the restriction to finite direct products, as the standard use of 'variety' in universal algebra does not include this requirement.)

Let Ξ be the countable alphabet $X = \{x_1, x_2, \ldots\}$. A *term* over Ξ is built from the letters by concatenation and application of a unary operation $v \mapsto v^\omega$. For example, $(x_1x_2)^\omega x_1$ is a term. We will interpret these terms in finite monoids in the obvious way, by considering a valuation $\psi : \Xi \to M$ and giving concatenation and the ω operator their usual meaning in M. For this reason, we do not distinguish between $(uv)w$ and $u(vw)$, where u, v and w are

themselves terms, nor between terms u^ω and $(u^\omega)^\omega$, as these will be equivalent under every valuation.

An *identity* is a formal equation $u = v$, where u and v are terms. We say that a monoid M *satisfies* the identity, and write $M \models (u = v)$, if u and v are equal under every valuation into M. The family of all finite monoids satisfying a given set of identities is a pseudovariety, and we say that the pseudovariety is *defined* by the set of identities. (We hasten to add that the identities we consider here are merely special instances of a much more general class of *pseudoidentities*. Under this broader definition, every pseudovariety is defined by a set of pseudoidentities. See, for instance, Almeida [1].)

We consider three particular pseudovarieties that will be of importance in this paper. First, the pseudovariety **Ap** consists of the *aperiodic* finite monoids, those that contain no nontrivial groups. **Ap** is defined by the identity $x_1^\omega = x_1 x_1^\omega$. If Σ is a finite alphabet and $L \subseteq \Sigma^*$ is a regular language, then $M(L) \in \mathbf{Ap}$ if and only if L is definable by a first-order sentence over $<$.

The pseudovariety **DA** is defined by the pair of identities

$$(x_1x_2x_3)^\omega x_2 (x_1x_2x_3)^\omega = (x_1x_2x_3)^\omega, x_1^\omega = x_1 x_1^\omega.$$

There are many equivalent characterizations of this pseudovariety in terms of other identities, the ideal structure of the monoids, and logic. For us the most important one is this: If $L \subseteq \Sigma^*$, then $M(L) \in \mathbf{DA}$ if and only if L is definable in $FO^2[<]$.

The pseudovariety **J** consists of finite monoids that satisfy the pair of identities

$$(x_1x_2)^\omega = (x_2x_1)^\omega, x_1^\omega = x_1 x_1^\omega.$$

This is equivalent to the identities

$$(x_1x_2)^\omega x_1 = x_2 (x_1x_2)^\omega = (x_1x_2)^\omega, x_1^\omega = x_1 x_1^\omega.$$

Alternatively, **J** consists of finite monoids M such that for all $s, t \in M$, $MsM = MtM$ implies $s = t$. Such monoids are said to be $\mathcal{J}$-trivial.

A theorem due to I. Simon [14] describes the regular languages whose syntactic monoids are in **J**. Let $w \in \Sigma^*$. We denote by $c(w)$ the *content* of w; that is, the set of letters of Σ that appear in w. We say that $v = \sigma_1 \cdots \sigma_k$, where each $\sigma_i \in \Sigma$, is a *subword* of w if

$$w = w_0 \sigma_1 w_1 \cdots \sigma_k w_k$$

for some $w_i \in \Sigma$. We denote by L_v the set of all words in Σ^* of which v is a subword. Let $k \geq 1$. We define an equivalence relation $\sim_k$ on Σ^* that identifies two words if and only if they contain the same subwords of length no more than k. (In particular, $w_1 \sim_1 w_2$ if and only if $c(w_1) = c(w_2)$.) Simon's theorem is:

▶ **Theorem 2.** *Let $L \subset \Sigma^*$ be a regular language. The following are equivalent:*

- *$M(L) \in \mathbf{J}$.*
- *L is a boolean combination of languages of the form L_u, with $u \in \Sigma^*$.*
- *L is a union of $\sim_k$-classes for some $k \geq 1$.*

The equivalence of the last two items is obvious. It is rather easy to show that $\Sigma^*/\sim_k \in \mathbf{J}$, and as a result the last two items imply that L is recognized by a monoid in **J**, and thus by Proposition 1, $M(L) \in \mathbf{J}$. The deep content of the theorem is that the first condition implies the others. The theorem can also be formulated in first-order logic: $M(L) \in \mathbf{J}$ if and only if L is defined by a boolean combination of Σ_1-sentences over $<$.

3 Two-sided Semidirect Products

In this section we describe an operation on both finite monoids and pseudovarieties, the *two-sided semidirect product.* This was given its formal description by Rhodes and Tilson [11], but it has precursors in automata theory in the work of Schützenberger on sequential bimachines [12], Krohn, Mateosian and Rhodes [8], and Eilenberg on triple products [3].

Let M and N be finite monoids. We will follow the standard practice of writing the product in N additively, and thus write its identity element as 0. This is not intended to suggest that N is commutative, but is simply a device for making the notation more readable. A *left action* of M on N is a mapping

$$(m, n) \mapsto mn \in N$$

from $M \times N$ into N that satisfies the axioms

$$\begin{aligned} m(n_1 + n_2) &= mn_1 + mn_2 \\ m_1(m_2 n) &= (m_1 m_2)n \\ m0 &= 0 \\ 1n &= n \end{aligned}$$

for all $m, m_1, m_2 \in M$, $n, n_1, n_2 \in N$.

A *right action* $(m, n) \mapsto nm$ of M on N is defined analogously. A *compatible pair* of actions consists of a left action and a right action of M on N that satisfy the additional axiom

$$m_1(nm_2) = (m_1 n)m_2,$$

for all $m_1, m_2 \in M$, $n \in N$. This justifies the notation $m_1 n m_2$ that we will henceforth use.

Given such a compatible pair, we define a monoid called the *two-sided semidirect product* $N * *M$. The underlying set is just the cartesian product $N \times M$, and the multiplication is given by

$$(n, m)(n', m') = (nm' + mn', mm').$$

It is straightforward to verify that this product is associative, and that $(0, 1)$ is the identity element.

Observe that the notation $N * *M$ suppresses mention of the action pair, so in fact there may be several non-isomorphic two-sided semidirect products of N and M. There is always at least one compatible action pair: these are the actions given by $mn = nm = n$ for all m, n. In this case, the resulting two-sided semidirect product reduces to the direct product. Moreover, there is always a compatible pair of actions of M on a direct product of $|M|^2$ copies of N. If we view the latter as the set of maps $F : M \times M \to N$ with componentwise multiplication, then the actions are given by

$$(mF)(m_1, m_2) = F(m_1, mm_2)$$

$$(Fm)(m_1, m_2) = F(m_1 m, m_2).$$

The resulting two-sided semidirect product is called the *block product* of N and M. This monoid has every two-sided semidirect product $N * *M$ as a divisor.

If $\mathbf{V}$ and $\mathbf{W}$ are pseudovarieties of finite monoids then we define $\mathbf{W} * *\mathbf{V}$ to be the collection of finite monoids that divide some two-sided semidirect product $N * *M$ with

$M \in \mathbf{V}$, $N \in \mathbf{W}$. $\mathbf{W} * *\mathbf{V}$ is itself a pseudovariety. We stress that this operation on pseudovarieties is not associative.

Throughout the proof of the main theorem we will use the following description of the regular languages recognized by members of $\mathbf{W} * *\mathbf{V}$. This is adapted from Thérien [18], and is a relatively straightforward translation from the definition of the product. Let $\alpha : \Sigma^* \to M$ be a homomorphism into a finite monoid, and let $\Gamma = M \times \Sigma \times M$. We view Γ as another alphabet. We define a length-preserving map τ_α (not a homomorphism) from Σ^* to Γ^* by

$$\tau_\alpha(\sigma_1 \cdots \sigma_k) = \gamma_1 \cdots \gamma_k,$$

where

$$\gamma_i = (\alpha(\sigma_1 \cdots \sigma_{i-1}), \sigma_i, \alpha(\sigma_{i+1} \cdots \sigma_k) \in \Gamma.$$

(If $i = 1$, we interpret the right-hand side as $(1, \sigma_1, \alpha(\sigma_2 \cdots \sigma_k))$, and similarly if $i = k$.)

▶ **Proposition 3.** *Let $L \subseteq \Sigma^*$ be a regular language. $M(L) \in \mathbf{W} * *\mathbf{V}$ if and only if there exist $M \in \mathbf{V}$, and a homomorphism $\alpha : \Sigma^* \to M$, such that L is a boolean combination of sets of the form*

$$\tau_\alpha^{-1}(K) \cap \alpha^{-1}(m),$$

where $m \in M$ and $K \subseteq \Gamma^*$ is recognized by a monoid in $\mathbf{W}$.

4 The Main Theorem

We define a sequence $\mathbf{V}_n$ of pseudovarieties of finite monoids as follows: $\mathbf{V}_1 = \mathbf{J}$, and, for $n \geq 1$, $\mathbf{V}_{n+1} = \mathbf{V}_n * *\mathbf{J}$.

▶ **Theorem 4.** *Let Σ be a finite alphabet, and let $L \subseteq \Sigma^*$. Let $n \geq 1$. $L \in FO_n^2[<]$ if and only if $M(L) \in \mathbf{V}_n$.*

The remainder of this section is devoted to the proof of Theorem 4.

We first prove that if L is recognized by a monoid in $\mathbf{V}_n$, then L is defined by a sentence of $FO_n^2[<]$. We show this by induction on n. For the case $n = 1$, Theorem 2 says that L is a finite boolean combination of languages of the form L_u, where $u \in \Sigma^*$. Each L_u is defined by a two-variable sentence with alternation depth 1 in an obvious way: For example, $L_{\sigma\tau\tau}$ is defined by the sentence

$$\exists x(Q_\sigma x \wedge \exists y(x < y \wedge Q_\tau y \wedge \exists x(y < x \wedge Q_\tau x))).$$

Now suppose $n > 1$. Then L is recognized by a monoid in $\mathbf{V}_{n-1} * *\mathbf{J}$. There are accordingly monoids $M \in \mathbf{J}$, $N \in \mathbf{V}_{n-1}$, and a morphism $\alpha : \Sigma^* \to M$ as in Proposition 3 above. We need to show that there is a formula of alternation depth no more than n defining each language of the form

$$\alpha^{-1}(m) \cap \tau_\alpha^{-1}(K),$$

where $m \in M$ and $K \subseteq \Gamma^* = (M \times \Sigma \times M)^*$ is recognized by N.

By Theorem 2, $\alpha^{-1}(m)$ is a boolean combination of languages of the form L_u, and so, as we saw above, is definable in alternation depth 1. So we turn to $\tau_\alpha^{-1}(K)$. By the inductive hypothesis, K is defined by a sentence ψ of alternation depth no more than $n-1$. The trick is to rewrite ψ to obtain a defining sentence for $\tau_\alpha^{-1}(K)$ while increasing the alternation depth by at most 1. This is accomplished simply by taking each of the atomic formulas $Q_{(m,\sigma,m')}x$ occurring in ψ and replacing it by a formula with x free and with quantifier depth 1. What should this formula say? It must assert that the letter in position x is σ, that the prefix

$v \in \Sigma^*$ of letters preceding this position satisfies $\alpha(v) = m$, and, similarly, that the suffix v' following this position satisfies $\alpha(v') = m'$. The first of these conditions is given by $Q_\sigma x$. The second, is, by Theorem 2, equivalent to a boolean combination of formulas asserting that v contains $\sigma_1, \ldots, \sigma_r$ as a subword, which is expressed by

$$\exists y(y < x \wedge Q_{\sigma_r} y \wedge \exists x(x < y \wedge Q_{\sigma_{r-1}} x \wedge \cdots)),$$

and the third by a boolean combination of analogous formulas. We accordingly replace $Q_{(m,\sigma,m')}x$ by a boolean combination of formulas with alternation depth no more than 1 to obtain the defining sentence for $\tau_\alpha^{-1}(K)$.

We now prove the converse: if L is defined by a sentence of $FO_n^2[<]$, then it is recognized by a monoid in $\mathbf{V}_n$.

Suppose $n > 0$, and let ϕ be a two-variable defining sentence for L. We write this in our standard form described earlier. Let us look at a quantified subformula ψ of ϕ that has quantifier alternation 1 and that is maximal for this property. We call ψ an *innermost block* of ϕ. In terms of the parse tree of ϕ, we are looking for nodes of minimal depth that are labeled by a quantifier, and such that every quantifier in the subtree rooted at this node is of the same type. The innermost blocks of ϕ are the formulas given by these subtrees.

If ϕ itself has quantifier alternation 1, then each innermost block ψ is a sentence, and ϕ is obtained from these blocks by disjunction and conjunction. Otherwise ψ has one free variable. Let's say this free variable is x. Suppose the quantifier in ψ is $\exists$. (If the quantifier in ψ is $\forall$ then its negation is a formula in which the only quantifier is $\exists$; we apply the transformations described below to this existential formula.) Since there are no negations in ψ we can, in the standard way (but introducing new variables in the process) rewrite ψ in prefix form as an ordinary Σ_1 formula

$$\exists y_1 \exists y_2 \cdots \exists y_r \theta(x, y_1, \ldots, y_r).$$

where θ is quantifier-free.

If $n = 1$ then the free variable x does not appear. Thus we can further rewrite ψ as a disjunction of sentences of the form

$$\exists y_1 \exists y_2 \cdots \exists y_r \Big(\bigwedge_{i=1}^{r} Q_{\sigma_i} y_i \wedge \rho(y_1, \ldots, y_r) \Big),$$

where ρ uniquely specifies the ordering among the y_i. (For example, with $r = 3$, ρ might have the form $y_1 = y_3 < y_2$.) Seen this way, ψ simply asserts the presence of certain subwords (and, had we begun with a universal quantifier, the absence of certain subwords.) In this case ϕ defines a boolean combination of languages of the form L_u, which by Theorem 2, is recognized by a monoid in $\mathbf{J}$. This is the base of our induction.

If $n > 1$, then we rewrite ψ as a disjunction of formulas of the form

$$\exists z_1 \cdots \exists z_t \exists z'_1 \cdots \exists z'_t \theta,$$

where θ is

$$\bigwedge_{i=1}^{t} (Q_{\sigma_i} z_i \wedge (z_i < x)) \wedge \rho_1(z_1, \cdots, z_t) \wedge \bigwedge_{j=1}^{t'} (Q_{\sigma'_j} z'_j \wedge (z'_j > x)) \wedge \rho_2(z'_1, \cdots, z'_t).$$

Let's denote this formula, which has x free, by

$$\zeta[x, u, v, \rho_1, \rho_2],$$

where $u = (\sigma_1, \ldots, \sigma_t)$ $v = (\sigma'_1, \ldots, \sigma'_{t'})$. If we started with an innermost block beginning with a universal quantifer, then this procedure produces the negation of a disjunction of these formulas. So we suppose ϕ has been transformed so that all its innermost blocks have been replaced by boolean combinations of such ζ.

Let s be the maximum of all the t, t' that occur in these formulas. Let $M = \Sigma^*/\sim_s$, as defined in Section 2. Let $\alpha : \Sigma^* \to M$ be the homomorphism that maps each word to its $\sim_s$-class. Recall that $M \in \mathbf{J}$.

We now rewrite ϕ and replace it by a new sentence ϕ' over the alphabet $\Gamma = M \times \Sigma \times M$. The idea is simply to express properties of a word $w \in \Sigma^*$ in terms of properties of $\tau_\alpha(w) \in \Gamma^*$ This is easy to do, because the two words have the same set of positions, and because the letters of $\tau_\alpha(w)$ encode additional information about each position. The subformula $\zeta[x, u, v, \rho_1, \rho_2]$ states that the prefix of w consisting of positions to the left of the position x contains a certain subword w_1 of length no more than s, and that the suffix consisting of positions to the right of x contains another such subword w_2. Equivalently, the letter of $\tau_\alpha(w)$ in position x is (m, σ, m'), where the $\sim_s$-class m contains w_1 as a subword, and the $\sim_s$-class m' contains w'_2. We thus replace each ζ by a disjunction of the atomic formulas $Q_{(m,\sigma,m')}x$ over all such m, m'. The result is that all the innermost blocks have now been eliminated and replaced by a boolean combination of these atomic formulas, which can in turn be written as a disjunction of such formulas.

We also replace each $Q_\sigma x$ that occurs outside an innermost block by the disjunction of the $Q_{(m,\sigma,m')}x$ over all $m, m' \in M$. The resulting sentence ϕ' is a two-variable sentence of quantifier depth $n-1$. Thus, by the induction hypothesis, the language $K \subseteq \Gamma^*$ defined by ϕ' is recognized by a monoid N in $\mathbf{V}_{n-1}$. We have constructed ϕ' so that $w \models \phi$ if and only if $\tau_\alpha(w) \models \phi'$. Thus, by Proposition 3, L is recognized by a monoid in $\mathbf{V}_{n-1} * *\mathbf{J} = \mathbf{V}_k$.

5 Strictness and Decidability

Here we use our main theorem to give a new proof that the alternation hierarchy is strict. This was first shown in [20]. We also discuss the question of decidability of the levels of the hierarchy.

We define two sequences of terms $\{u_n\}_{n\geq 1}$, $\{v_n\}_{n\geq 1}$ as follows.

$$u_1 = (x_1x_2)^\omega, v_1 = (x_2x_1)^\omega.$$

If $n \geq 1$, we set

$$u_{n+1} = (x_1 \cdots x_{2n}x_{2n+1})^\omega u_n (x_{2n+2}x_1 \cdots x_{2n})^\omega$$

$$v_{n+1} = (x_1 \cdots x_{2n}x_{2n+1})^\omega v_n (x_{2n+2}x_1 \cdots x_{2n})^\omega$$

▶ **Proposition 5.** *Let $n \geq 1$. If $M \in \mathbf{V}_n$, then $M \models (u_n = v_n)$, and $M \models (x_1^\omega = x_1x_1^\omega)$.*

Proof. For $n = 1$, the Proposition follows from the identities defining $\mathbf{J}$ that were given in Section 2. For the inductive step we will make repeated use of the following identities that also hold in $\mathbf{J}$, and that are direct consequences of the ones we gave earlier:

$$(x_1x_2x_3)^\omega x_2 = (x_1x_2x_3)^\omega = x_2(x_1x_2x_3)^\omega \tag{1}$$

Suppose then that $n \geq 1$, and that the Proposition holds for n. It is well known—and in any case follows easily from the kind of argument we give below—that the two-sided

semidirect product preserves aperiodicity. So we will only show $M \models (u_{n+1} = v_{n+1})$ for $M \in \mathbf{V}_{n+1}$. Since satisfaction of identities is preserved under division, we only need to show this in the case where M is a two-sided semidirect product $T * *K$, with $T \in \mathbf{V}_n$ and $K \in \mathbf{J}$. Consider a map ϕ from the set $\{x_1, x_2, \ldots\}$ into M with

$$\phi(x_i) = m_i = (t_i, k_i) \in T * *K$$

for all $i \geq 1$. Now suppose $x_{i_1} \cdots x_{i_p}$ is a term formed just by concatenating variables (*i.e.*, without using ω). Then

$$\phi(x_{i_1} \cdots x_{i_p}) = \left(\sum_{j=1}^{p} k_{i_1} \cdots k_{i_{j-1}} t_{i_j} k_{i_{j+1}} \cdots k_{i_p}, k_{i_1} \cdots k_{i_p} \right). \tag{2}$$

There is an integer q such that $m^\omega = m^q$ for all $m \in T, K$ or M. Thus

$$\begin{aligned} \phi(u_{n+1}) &= \phi((x_1 x_2 \cdots x_{2n+1})^q u_n (x_{2n+2} x_1 \cdots x_{2n})^q) \\ &= (t, \gamma(u_n)), \end{aligned}$$

where t is a sum of the form displayed in Equation 2, and $\gamma(x_i) = k_i$ for all i. Let us analyze the summands of t. Let $s = q \cdot (2n+1)$. If $j \leq s$, then the j^{th} summand is

$$k_{i_j} = k_{i_1} \cdots k_{i_{j-1}} t_{i_j} k_R,$$

where

$$k_R = (k_{2n+2} k_1 \cdots k_{2n})^q.$$

This follows from the absorbing property given in Equation 1 of the $\mathcal{J}$-trivial monoid K. Similarly, if $s \geq p - j$, the j^{th} summand is

$$k_L t_{i_j} k_{i_{j+1}} \cdots k_{i_p},$$

where

$$k_L = (k_1 \cdots k_{2n} k_{2n+1})^q.$$

If $s < j < n - s$, then the j^{th} summand is $k_L t_{i_j} k_R$, so that the sum of these middle terms is

$$\sum_{j=s+1}^{n-s} k_L t_{i_j} k_R = k_L \left(\sum_{j=s+1}^{n-s} t_{i_j} \right) k_R = k_L \psi(u_n) k_R,$$

where $\psi(x_i) = t_i$ for all i. We thus can write $\phi(u_{n+1})$ in the form

$$\phi(u_{n+1}) = (t_L + k_L \psi(u_n) k_R + t_R, \gamma(u_{n+1})).$$

When we compute $\phi(v_{n+1})$, the values of t_L and t_R are unchanged, and we have

$$\phi(v_{n+1}) = (t_L + k_L \psi(v_n) k_R + t_R, \gamma(v_{n+1})).$$

From the identities for $\mathbf{J}$ in Equation 1 we have $K \models (u_{n+1} = v_{n+1})$, and thus $\gamma(u_{n+1}) = \gamma(v_{n+1})$. From the inductive hypothesis we have $T \models (u_n = v_n)$, so $\psi(u_n) = \psi(v_n)$, and thus $\phi(u_{n+1}) = \phi(v_{n+1})$, as required.

◀

We will use these identities to show that the alternation hierarchy is strict. We begin by defining, for each finite alphabet Σ, an equivalence relation $\equiv_\Sigma$ on Σ^*. (In fact, $\equiv_\Sigma$ will be a congruence on Σ^* whose quotient is the *free idempotent monoid* on Σ. This construction is very well known; see, for, example, Eilenberg [3].)

If $|\Sigma| = 1$, then $\equiv_\Sigma$ identifies two distinct words over Σ if and only if they are both nonempty (so that there are two equivalence classes, one containing the empty word, and the other containing all the nonempty words). Now suppose $|\Sigma| > 1$, and that $\equiv_\Gamma$ has been defined for all proper subalphabets Γ of Σ. Let $w_1, w_2 \in \Sigma^*$. If the set of distinct letters $\Gamma = c(w_1)$ appearing in w_1 is a proper subset Γ of Σ, then we set $w_1 \equiv_\Sigma w_2$ if and only if $c(w_2) = \Gamma$, and $w_1 \equiv_\Gamma w_2$. Otherwise, $c(w_1) = c(w_2) = \Sigma$. Let u_i denote the maximal prefix of w_i such that $c(u_i) \neq \Sigma$, and similarly let v_i denote the maximal suffix of w_i such that $c(v_i) \neq \Sigma$. We can then write

$$w_i = u_i\sigma_i y_i = z_i\tau_i v_i,$$

where $\sigma_i, \tau_i \in \Sigma$. We define $w_1 \equiv_\Sigma w_2$ if and only if $\sigma_1 = \sigma_2$, $\tau_1 = \tau_2$, $u_1 \equiv_{c(u_1)} u_2$, and $v_1 \equiv_{c(v_1)} v_2$.

Easily, $\equiv_\Sigma$ is a congruence of finite index on Σ^*. We denote the $\equiv_\Sigma$-class of $w \in \Sigma^*$ by $[w]_\equiv$. The language $[w]_\equiv$ is regular; moreover, for every word $u \in \Sigma^*$, $u \equiv_\Sigma u^2$, which implies that $m^2 = m$, or, equivalently $m^\omega = m$, for every $m \in M([w]_\Sigma)$.

▶ **Lemma 6.** *Let $|\Sigma| = n$. Every class of $\equiv_\Sigma$ is definable by a sentence of $FO^2_n[<]$.*

Proof. We prove this by induction on n. For $n = 1$, we have $\Sigma = \{\sigma\}$, and the two classes are defined by the sentences

$$\exists x Q_\sigma x$$

and

$$\forall x(x < x).$$

(Note that we allow our formulas to be interpreted in the empty word, which satisfies every universally quantified sentence.)

Assume now that $n > 1$, and that the claim is true for all subalphabets of Σ. Let $w \in \Sigma^*$. If $c(w) \neq \Sigma$, then $[w]_\Sigma = [w]_\Gamma$ for some proper subalphabet Γ of Σ. The inductive hypothesis implies that this class is defined by a sentence of $FO^2_{n-1}[<]$. So we assume $c(w) = \Sigma$, and write $w = u\sigma x = y\tau v$, where u, v are, respectively, the maximal prefix and suffix of w that do not contain all the letters of Σ. To express the property that every letter except σ occurs in the prefix w we use the sentence

$$\exists x(Q_\sigma x \wedge \bigwedge_{\sigma' \neq \sigma} \exists y(y < x \wedge Q_{\sigma'} y) \wedge \forall y(y \geq x \vee \bigvee_{\sigma' \neq \sigma} Q_{\sigma'} y))).$$

Note that this sentence has alternation depth $2 \leq n$.

To express the property that the prefix preceding the first position containing σ belongs to a particular $\equiv_\Gamma$-class, where $\Gamma = c(u)$, we apply the inductive hypothesis: There is a sentence ψ of alternation depth less than n defining $[u]_\Gamma$. We modify ψ by replacing each existentially quantified subformula $\exists x\zeta$ by

$$\exists x(\zeta \wedge \forall y(y \geq x \vee \bigvee_{\sigma' \neq \sigma} Q_{\sigma'} y)),$$

and each universally quantified subformula $\forall x\zeta$ by

$$\forall x(\zeta \vee \exists y(y \leq x \wedge Q_\sigma y)).$$

The resulting sentence has alternation depth no more than n and defines the set of strings such that the maximal prefix that does not contain σ is in $[u]_\Gamma$. The conjunction of these two sentences, along with the analogues for the suffix, defines $[w]_\Sigma$.

◀

▶ **Lemma 7.** *Let $n \geq 1$, and let $|\Sigma| = 2n$. There is a word $w \in \Sigma^*$ such that $M([w]_\Sigma)$ does not satisfy $(u_n = v_n)$.*

Proof. Let u'_n and v'_n be the terms that result from removing all occurrences of the operator ω from u_n and v_n, respectively. Let $\Sigma = \{\sigma_1, \cdots, \sigma_{2n}\}$, and let $w_1^{(n)}, w_2^{(n)} \in \Sigma^*$ be the respective words that result when each occurrence of a variable x_i in u_n or v_n is replaced by σ_i. It is enough to show that $w_1^{(n)} \not\equiv_\Sigma w_2^{(n)}$. The reason is this: We can take $M = M([w_1^{(n)}]_\Sigma)$. Since $M \models (x^\omega = x)$, if we had $M \models (u_n = v_n)$ then $M \models (u'_n = v'_n)$. But that case we would obtain $w_1^{(n)} \equiv_\Sigma w_2^{(n)}$.

We prove that $w_1^{(n)} \not\equiv_\Sigma w_2^{(n)}$ by induction on n. For $n = 1$ we have

$$w_1^{(1)} = \sigma_1\sigma_2 \not\equiv_\Sigma \sigma_2\sigma_1 = w_2^{(1)}.$$

For $n > 1$ we have

$$w_j^{(n)} = \sigma_1 \cdots \sigma_{2n-1} w_j^{(n-1)} \sigma_{2n}\sigma_1 \cdots \sigma_{2n-2},$$

for $j = 1, 2$. The maximal prefix of $w_j^{(n)}$ that does not contain all the letters of Σ is $z_j = \sigma_1 \cdots \sigma_{2n-1} w_j^{(n-1)}$, and the maximal suffix of z_j not containing all the letters of $c(z_j)$ is $w_j^{(n-1)}$. By the inductive hypothesis, $w_1^{(n-1)} \not\equiv_\Gamma w_2^{(n-1)}$, where $\Gamma = \{\sigma_1, \ldots, \sigma_{2n-2}\}$, so we cannot have $w_1^{(n)} \equiv_\Sigma w_2^{(n)}$.

◀

We get the strictness of the hierarchy as a consequence of these two lemmas:

▶ **Theorem 8.** *For every $n > 1$ there is a language definable in $FO^2_n[<]$ that is not definable in $FO^2_{n-1}[<]$.*

Proof. For every $n > 1$, we must have $\mathbf{V}_{n-1} \subsetneq \mathbf{V}_n$, since equality at one level would imply equality at all higher levels, and we would have, in particular, $\mathbf{V}_n = \mathbf{V}_{2n}$ for some $n \geq 1$. But the two Lemmas, coupled with Theorem 4 and Proposition 5, provide an example of a language whose syntactic monoid is in $\mathbf{V}_{2n} \backslash \mathbf{V}_n$. Thus $\mathbf{V}_{n-1} \subsetneq \mathbf{V}_n$. Since every pseudovariety is generated by the syntactic monoids it contains, Theorem 4 gives the result. ◀

We now discuss the problem of *decidabilty:* Suppose we are given a regular language $L \subseteq \Sigma^*$, either by an automaton that recognizes L, or in terms of some other representation, such as a regular expression, from which we can effectively construct an automaton. Is there an algorithm for determining whether L can be defined by a sentence of $FO^2_n[<]$ for a fixed n? Of course, this begs the question of whether L can be defined by a sentence of $FO^2[<]$ at all, but this problem is solved by earlier work: Compute the syntactic monoid of $M(L)$ and determine whether $M(L)$ is in **DA**, by verifying the identities for **DA**. (Note that in verifying the identities in a particular monoid M, the symbol ω in these identities can be replaced by $|M|$.)

Algebraic methods provide a powerful tool for answering such decision questions (and, more generally, for proving that a given language cannot be defined in a logic, as we did in Lemma 7 above), since the multiplication table of the syntactic monoid of L can be effectively computed from any reasonable representation of L. However, in order to apply this method

here, we need an algorithm for determining whether a given finite monoid belongs to $\mathbf{V}_n$, for any given n. Identities defining these pseudovarieties would provide us with precisely such an algorithm, but the identities $(u_n = v_n, x_1^\omega = x_1 x_1^\omega)$ that we have exhibited have only been proved to be necessary conditions for membership in $\mathbf{V}_n$.

In fact, these identities are from a paper by by Almeida and Weil [2], where they appear as part of a general scheme for obtaining identities for pseudovarieties of the form $\mathbf{V} * *\mathbf{J}$ when $\mathbf{V} \subseteq \mathbf{DA}$ and identities for $\mathbf{V}$ are known. The result stated there would imply that satisfaction of $(u_n = v_n, x_1^\omega = x_1 x_1^\omega)$ is also a sufficient condition for membership in $\mathbf{V}_n$, and thus resolve the decidability question. However, this paper is known to contain an error. A second paper, by Weil [19], explains the nature of the problem: The proof that the identities are sufficient requires a particular finite rank property for categories that are globally covered by members of $\mathbf{V}_{n-1}$. (Even defining these terms would take us too far afield; the interested reader is referred to [2] and [19] and the many references given there.)

As we have already mentioned, for $n = 1$ the identities $(u_n = v_n, x_1^\omega = x_1 x_1^\omega)$ are known to define $\mathbf{J}$. The finite rank property, thanks to a theorem of Knast [7], is known to hold for $\mathbf{J}$, and therefore the identities for $n = 2$ do indeed define $\mathbf{J} * *\mathbf{J}$. As a consequence, we have:

▶ **Theorem 9.** *It is decidable whether a given regular language is definable in* $FO_1^2[<]$, *or in* $FO_2^2[<]$.

. For level 1, the answer is again membership of the syntactic monoid in $\mathbf{J}$; for the second level the answer is unknown. We suspect that the problem of alternation depth in $FO^2[<]$, while still challenging, will turn out to be easier.

The finite rank property is not known to hold for $\mathbf{V}_{n-1}$, if $n > 2$. Thus the decidability problem remains open for higher levels of the hierarchy. This does not rule out the possibility that the identities might be proved sufficient even without the assumption of finite rank.

▶ **Conjecture 10.** *Let* $n \geq 1$. $M \in \mathbf{V}_n$ *if and only if* $M \models (u_n = v_n)$, *and* $M \models (x_1^\omega = x_1 x_1^\omega)$. *In particular, it is decidable whether a given regular language is definable in* $FO_n^2[<]$.

. For level 1, the answer is again membership of the syntactic monoid in $\mathbf{J}$; for the second level the answer is unknown. We suspect that the problem of alternation depth in $FO^2[<]$, while still challenging, will turn out to be easier.

Kufleitner and Weil [9] also study the alternation hierarchy algebraically, and introduce a very different-looking sequence of pseudovarieties $\mathbf{W}_n$ with the property that the syntactic monoid of every language with alternation depth exactly n is between $\mathbf{W}_n$ and $\mathbf{W}_{n+1}$. These pseudovarieties are known to have decidable membership problems. Kufleitner and Weil conjecture that in fact this sequence of pseudovarieties exactly captures the alternation hierarchy. This conjecture would settle the decision problem for alternation depth (and would also, coupled with our results, imply $\mathbf{W}_n = \mathbf{V}_n$ for all n.) It would be interesting to try to establish containments between $\mathbf{W}_n$ and $\mathbf{V}_n$.

Finally, we mention the similar-looking problem of *dot-depth.* Full first-order logic over $<$ interpreted in finite words defines all the languages with syntactic monoids in $\mathbf{Ap}$. The problem of determining the exact alternation depth of a language in this setting–the problem of calculating the so-called *dot-depth* of a language–has been open for nearly forty years. For level 1, the answer is again membership of the syntactic monoid in $\mathbf{J}$; for the second level the answer is unknown. We suspect that the problem of alternation depth in $FO^2[<]$, while still challenging, will turn out to be easier.

Acknowledgements. Many thanks to Phillip Weis, Neil Immerman, Jorge Almeida and Pascal Weil for sharing and discussing their work with me.

References

1. J. Almeida, *Finite Semigroups and Universal Algebra,* World Scientific, Singapore, 1994.
2. J. Almeida, P. Weil. "Profinite Categories and Semidirect Products, *J. Pure and Applied Algebra* **123** (1998) 1 - 50.
3. S. Eilenberg, *Automata, Languages and Machines,* vol. B, Academic Press, New York, 1976.
4. K. Etessami, M. Vardi, and T. Wilke, "First-Order Logic with Two Variables and Unary Temporal Logic", *Proceedings, 12th IEEE Symposium on Logic in Computer Science,* 228-235 (1996).
5. N. Immerman and D. Kozen, "Definability with a Bounded Number of Bound Variables", *Information and Computation,* **83**, 121-139 (1989).
6. J. Kamp, *Tense Logic and the Theory of Linear Order,* Ph.D. thesis, UCLA (1968).
7. R. Knast, "Some Theorems on Graph Congruences", *Informatique Théorique et Applications,* 331-342 (1983).
8. K. Krohn, R. Mateosian, and J. Rhodes, "Methods of the Algebraic Theory of Machines. I: Decomposition Theorem for Generalized Machines: Properties Preserved under Series and Parallel Compositions of Machines", *J. Comput. Syst. Sci,* 55-85 (1967).
9. M. Kufleitner, P. Weil, "On FO2 Quantifier Alternation over Words", MFCS 2009, Lecture Notes in Computer Science **5734** (Springer, 2009), pp. 513-524.
10. J. E. Pin, *Varieties of Formal Languages,* Plenum, London, 1986.
11. J. Rhodes and B. Tilson, "The Kernel of Monoid Morphisms", *J. Pure and Applied Algebra* **62** (1989) 227–268.
12. "A remark on finite transducers", *Information and Control* **4** (1961), 185-196.
13. T. Schwentick, D. Thérien and H. Vollmer. "Partially-ordered Two-way Automata: A New Characterization of **DA**', In *Proc. 5th Developments in Language Theory (DLT 2001),* (2001) 239-250.
14. I. Simon. "Piecewise testable events", in *Automata Theory and Formal Languages,* (1975), 214-222.
15. H. Straubing, *Finite Automata, Formal Logic and Circuit Complexity,* Birkhäuser, Boston, 1994.
16. H. Straubing and D. Thérien, "Weakly Iterated Block Products of Finite Monoids", *LATIN 2002, Lecture Notes in Computer Science,* **2286** (Springer, 2002),91-104.
17. P. Tesson and D. Thérien, "Diamonds are Forever: The Variety **DA**", in *Semigroups, Algorithms, Automata and Languages*, World Scientific, Singapore (2002), 475-500.
18. D. Thérien, "Two-sided wreath products of categories", *J. Pure and Applied Algebra* **74** (1991) 307-315.
19. P. Weil. "Profinite Methods in Semigroup Theory, *Intern. J. Algebra and Computation* **12** (2002) 137-178.
20. P. Weis and N. Immerman, "Structure Theorem and Strict Alternation Hierarchy for FO^2 on Words", *Logical Methods in Computer Science,* **5** (2009).

Non-Commutative Infinitary Peano Arithmetic

Makoto Tatsuta[1] and Stefano Berardi[2]

1 **National Institute of Informatics**
2-1-2 Hitotsubashi, Tokyo 101-8430, Japan
tatsuta@nii.ac.jp

2 **Department of Computer Science, Torino University**
corso Svizzera 185, 10149 Torino, Italy
stefano@di.unito.it

Abstract

Does there exist any sequent calculus such that it is a subclassical logic and it becomes classical logic when the exchange rules are added? The first contribution of this paper is answering this question for infinitary Peano arithmetic. This paper defines infinitary Peano arithmetic with non-commutative sequents, called non-commutative infinitary Peano arithmetic, so that the system becomes equivalent to Peano arithmetic with the omega-rule if the the exchange rule is added to this system. This system is unique among other non-commutative systems, since all the logical connectives have standard meaning and specifically the commutativity for conjunction and disjunction is derivable. This paper shows that the provability in non-commutative infinitary Peano arithmetic is equivalent to Heyting arithmetic with the recursive omega rule and the law of excluded middle for Sigma-0-1 formulas. Thus, non-commutative infinitary Peano arithmetic is shown to be a subclassical logic. The cut elimination theorem in this system is also proved. The second contribution of this paper is introducing infinitary Peano arithmetic having antecedent-grouping and no right exchange rules. The first contribution of this paper is achieved through this system. This system is obtained from the positive fragment of infinitary Peano arithmetic without the exchange rules by extending it from a positive fragment to a full system, preserving its 1-backtracking game semantics. This paper shows that this system is equivalent to both non-commutative infinitary Peano arithmetic, and Heyting arithmetic with the recursive omega rule and the Sigma-0-1 excluded middle.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Proof theory, cut elimination, intuitionistic logic, subclassical logic, infinitary logic, recursive omega rules, substructural logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.538

1 Introduction

Substructural logics, which are logical systems without some of the contraction rule, the weakening rule, and the exchange rule, have been actively studied in both mathematical logic and computer science. For example, linear logic, which is a logical system without the contraction rule or the weakening rule is successful [9].

Does there exist any sequent calculus such that it is a subclassical logic and it becomes classical logic when the exchange rules are added? The first contribution of this paper is answering this question for infinitary Peano arithmetic. This paper defines infinitary Peano arithmetic with non-commutative sequents, called non-commutative infinitary Peano arithmetic, so that the system becomes equivalent to Peano arithmetic with the omega-rules if the exchange rules are added to this system. This paper shows that the provability in non-commutative infinitary Peano arithmetic is equivalent to Heyting arithmetic with

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 538–552

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

the recursive omega rules and the law of excluded middle for Σ^0_1 formulas. Thus, non-commutative infinitary Peano arithmetic is shown to be a subclassical logic.

Arithmetic without the exchange rule has not been studied yet. For infinitary arithmetic without the exchange rules, only its positive fragment was investigated in [4, 6, 2]. For a full system without the exchange rules, only the classical sequent calculus without the exchange rules is studied [13].

The second contribution of this paper is introducing infinitary Peano arithmetic having antecedent-grouping and no right exchange rules. The first contribution is achieved through this system. This system is obtained from the positive fragment of infinitary Peano arithmetic without the exchange rules by extending it from a positive fragment to a full system, preserving its 1-backtracking game semantics. This paper shows that this system is equivalent to both non-commutative infinitary Peano arithmetic, and Heyting arithmetic with the recursive omega rules and the Σ^0_1 excluded middle.

This paper will define non-commutative infinitary Peano arithmetic NCIPA as well as the arithmetic IPA^- having antecedent-grouping and no right exchange rules, and prove (1) NCIPA becomes equivalent to Peano arithmetic IPA with the ω-rules when the exchange rules are added to the system, (2) NCIPA is equivalent to Heyting arithmetic with the recursive ω-rules, called IHA, and the law EM_1 of excluded middle for Σ^0_1 formulas, (3) the cut elimination theorem in NCIPA, (4) the cut elimination theorem in IPA^-, and (5) translations between NCIPA and IPA^-.

IPA^- was inspired by 1-backtracking game semantics [5, 8]. [4] proved correspondence between its positive fragment and 1-backtracking game, by which a winning strategy corresponds to a proof. [2] also defined a sound and complete semantics for the fragment using interactive realizers.

IPA^- in this paper is a full system obtained from the positive fragment by adding implication. IPA^- is described by using a sequent $\Gamma \vdash \Delta$ with antecedent-grouping where formulas in the antecedent Γ are grouped and structural rules can be used only inside a group. We can also use the weakening rule and the contraction rule in the succedent Δ, but cannot use the exchange rule.

EM_1 is the principle $\forall x_1 \dots x_n (A \vee \neg A)$ for a Σ^0_1 formula A. This principle gives logical systems between intuitionistic logic and classical logic, which have been studied actively, in particular, for hidden algorithms in their proofs [1, 3, 11] and for their relation with continuation-passing style programs [7].

We design NCIPA from IPA^- so that it is based on ordinary sequents without antecedent-grouping, and the grouping information is represented by the length of a sequence of formulas. The translations between NCIPA and IPA^- will be defined so that they map the length of a sequence of formulas and the grouping information into each other. The equivalence between NCIPA and $\mathrm{IHA} + \mathrm{EM}_1$ is proved from the translations between NCIPA and IPA^-, and the equivalence between IPA^- and $\mathrm{IHA} + \mathrm{EM}_1$.

The implication from provability in $\mathrm{IHA} + \mathrm{EM}_1$ to provability in IPA^- is proved by using the cut elimination theorem in IPA^-.

The implication from provability in IPA^- to provability in $\mathrm{IHA} + \mathrm{EM}_1$ is proved by using flag formulas. A flag formula is a Π^0_1 formula and is defined for each formula in the succedent when its proof is given. Given a proof of $\Gamma \vdash A_1, \dots, A_n$, if the flag formula F_i of A_i is true, then every succedent in the proof is of length more than or equal to i. Flag formulas enable us to find the minimum length of the succedents in a proof even if the proof is infinite. The key idea is case analysis by a flag formula, which we can use since EM_1 proves $F_i \vee \neg F_i$.

The cut elimination theorem in NCIPA is proved by the translations between NCIPA and IPA^- and the cut elimination theorem in IPA^-.

A potential application of the equivalence results is program extraction with the halting problem oracle. Since NCIPA and IPA^- are equivalent to $\mathrm{IHA} + \mathrm{EM}_1$ and EM_1 corresponds to the halting problem oracle, we can extract a program with the halting problem oracle

from a proof in NCIPA or IPA$^-$. This program can be interpreted as a learning algorithm, using 1-backtracking and learning in the limit [7].

Section 2 defines infinitary arithmetic IPA and IHA. Section 3 presents IPA$^-$, and its cut elimination theorem is proved in Section 4. Section 5 proves the implication from IHA$+$EM$_1$ to IPA$^-$, and Section 6 proves the other implication from IPA$^-$ to IHA $+$ EM$_1$. Section 7 defines NCIPA, and shows the equivalence between NCIPA with the exchange rules and IPA. Section 8 gives the translations between NCIPA and IPA$^-$, and shows the equivalence between NCIPA and IHA $+$ EM$_1$. The cut elimination theorem in NCIPA is proved in Section 9.

2 Infinitary Arithmetic

We define the system IPA. It is Peano arithmetic based on infinitary logic where the inference rules $(\forall R)$ and $(\exists L)$ are replaced by the ω-rules with countably many assumptions, and it does not have induction rules. The induction principles are derivable.

▶ Definition 2.1 (language). The language of IPA is a first-order language generated from the following symbols: We have variables $x, y, z, \ldots$. Constants are numerals $0, 1, 2, \ldots$, denoted by $n, m, i, j, \ldots$.

Function symbols are denoted by $f, g, \ldots$. We assume that the set of function symbols is recursive, and the set of functions represented by function symbols is the same as the set of primitive recursive functions.

Terms are denoted by $s, t, \ldots$.

Predicate symbols are denoted by $P, Q, \ldots$. We assume that the set of predicate symbols is recursive, and the set of predicates represented by predicate symbols is the same as the set of primitive recursive predicates. We have 0-ary predicate symbols $\top$ and $\bot$, which mean the truth and the falsity respectively.

Formulas are defined by $A, B, C, \ldots ::= P(t_1, \ldots, t_n) | A \wedge B | A \vee B | A \to B | \forall x A | \exists x A$, where P is a predicate symbol of arity n. We will write $\neg A$ for $A \to \bot$.

A sentence is a closed formula. A sequence $A_1, \ldots, A_n$ $(n \geq 0)$ of sentences is denoted by $\Gamma, \Delta, \Pi, \Sigma, \ldots$. $|\Gamma|$ denotes its length. A^n denotes $A, \ldots, A$ (n times). $A[t/x]$ is the formula obtained from A by replacing x by t.

Sequents are of the form $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$ $(n, m \geq 0)$ where A_i, B_i sentences. We respect order of sentences in a sequence and a sequent.

IPA is based on infinitary logic where assumptions can be countably many. A proof in this system is defined as a well-founded recursive tree by inference rules.

We have the following inference rules given in Figure 1. In the rules $(Ax\ R)$ and $(Ax\ L)$, true and false refer to the truth value in the standard model. The rules $(\forall R)$ and $(\exists L)$ denote an inference of its conclusion from some recursive function f such that $f(m)$ is the code of a proof of the m-th assumption, for example, $\Gamma \vdash \Delta, A[m/x]$ for $(\forall R)$.

We give an accurate definition of a proof inductively as follows: $\lceil \cdot \rceil$ is a standard coding function and $\lceil e \rceil$ is a code of a syntactical object e. (1) For an inference rule except $(\forall R)$ or $(\exists L)$, $(\lceil L \rceil, \lceil S \rceil, P_1, \ldots, P_n)$ is a proof of the sequent S if its name is L, its instance is the inference of S from $S_1 \ldots S_n$, and P_i is a proof of S_i for $1 \leq i \leq n$. (2) For an inference rule among $(\forall R)$ and $(\exists L)$, $(\lceil L \rceil, \lceil S \rceil, f)$ is a proof of the sequent S if its name is L and its instance is the inference of S from $S_1[n/x]$ (for all n) and f is a recursive function such that $f(n)$ is the code of a proof of $S_1[n/x]$.

We will write $\Gamma \vdash_{\mathrm{IPA}} \Delta$ to denote that the sequent $\Gamma \vdash \Delta$ is provable in IPA. We will also use $\vdash_T$ for some other systems T we will introduce later.

We define the system IHA. It is Heyting arithmetic based on infinitary logic where the inference rules $(\forall R)$ and $(\exists L)$ are replaced by the recursive ω-rules.

The language is the same as that of IPA except that its sequents are intuitionistic sequents $A_1, \ldots, A_n \vdash B$ or $A_1, \ldots, A_n \vdash$.

$$\frac{}{\Gamma, A \vdash \Delta}\ (Ax\ L)\ (A \quad \text{a false atomic formula})$$

$$\frac{}{\Gamma \vdash \Delta, A}\ (Ax\ R)\ (A \quad \text{a true atomic formula})$$

$$\frac{\Gamma \vdash \Delta, A \quad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B}\ (\wedge R) \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ (\wedge L1) \qquad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ (\wedge L2)$$

$$\frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, A \vee B}\ (\vee R1) \qquad \frac{\Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \vee B}\ (\vee R2) \qquad \frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\ (\vee L)$$

$$\frac{\Gamma, A \vdash \Delta, B}{\Gamma \vdash \Delta, A \to B}\ (\to R) \qquad \frac{\Gamma \vdash \Delta, A \quad \Gamma, B \vdash \Sigma}{\Gamma, A \to B \vdash \Delta, \Sigma}\ (\to L)$$

$$\frac{\Gamma \vdash \Delta, A[m/x] \quad (\text{for all } m)}{\Gamma \vdash \Delta, \forall x A}\ (\forall R) \qquad \frac{\Gamma, A[m/x] \vdash \Delta}{\Gamma, \forall x A \vdash \Delta}\ (\forall L) \qquad \frac{\Gamma \vdash \Delta, A[m/x]}{\Gamma \vdash \Delta, \exists x A}\ (\exists R)$$

$$\frac{\Gamma, A[m/x] \vdash \Delta \quad (\text{for all } m)}{\Gamma, \exists x A \vdash \Delta}\ (\exists L) \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}\ (weak\ R) \qquad \frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\ (weak\ L)$$

$$\frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}\ (cont\ R) \qquad \frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\ (cont\ L)$$

$$\frac{\Gamma \vdash \Delta_1, B, A, \Delta_2}{\Gamma \vdash \Delta_1, A, B, \Delta_2}\ (exch\ R) \qquad \frac{\Gamma_1, B, A, \Gamma_2 \vdash \Delta}{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}\ (exch\ L)$$

Figure 1 Inference Rules of IPA

The inference rules are the same as those of IPA except that their sequents are restricted to intuitionistic sequents.

The law EM_1 of excluded middle for Σ^0_1 formulas is defined as the axiom schema $\forall x_1 \dots x_n (A \vee \neg A)$ for a Σ^0_1 formula A. This is a weaker version of the law of excluded middle. The system $\mathrm{IHA} + \mathrm{EM}_1$ strictly includes IHA and is strictly included in IPA.

Note that the identity rule $\Gamma, A \vdash A$ is provable. It is shown by induction on A.

3 The system IPA$^-$

We define the logical system IPA$^-$ of Peano arithmetic having the recursive ω-rules, antecedent-grouping, and no right exchange rules.

The language of IPA$^-$ is the same as that of IPA except that its sequents are different. A sequent in IPA$^-$ is of the form $\Gamma \vdash A_1, \dots, A_n$ where Γ is a sequence of sentences and n symbols of the symbol $-$. An example of the sequent is $A_1, -, A_2, A_3, -, A_4, -, A_5, A_6 \vdash B_1, B_2, B_3$.

In the sequent $\Gamma_0, -, \Gamma_1, -, \Gamma_2, \dots, -, \Gamma_n \vdash A_1, \dots, A_n$ where Γ_i is a sequence of sentences and does not contain the symbol $-$, the group Γ_0 means an initial group, and the group Γ_i corresponds to A_i.

$\Gamma, \Delta, \dots$ denote a sequence of both sentences and symbols $-$. We will write $-^n$ for $-, \dots, -$ (n times).

We have the following inference rules given in Figure 2

A proof in this system is defined as a well-founded recursive tree in a similar way to IPA. A proof of a formula A means a proof of the sequent $- \vdash A$.

Intuitive meaning of provable sequents is given by using the familiar interpretation of

$$\frac{}{\Gamma, A \vdash \Delta}\ (Ax\ L)\ (A \quad \text{a false atomic formula})$$

$$\frac{}{\Gamma \vdash \Delta, A}\ (Ax\ R)\ (A \quad \text{a true atomic formula})$$

$$\frac{\Gamma, - \vdash \Delta, A \wedge B, A \quad \Gamma, - \vdash \Delta, A \wedge B, B}{\Gamma \vdash \Delta, A \wedge B}\ (\wedge R)$$

$$\frac{\Gamma_1, A \wedge B, \Gamma_2, A \vdash \Delta}{\Gamma_1, A \wedge B, \Gamma_2 \vdash \Delta}\ (\wedge L1) \qquad \frac{\Gamma_1, A \wedge B, \Gamma_2, B \vdash \Delta}{\Gamma_1, A \wedge B, \Gamma_2 \vdash \Delta}\ (\wedge L2)$$

$$\frac{\Gamma, - \vdash \Delta, A \vee B, A}{\Gamma \vdash \Delta, A \vee B}\ (\vee R1) \qquad \frac{\Gamma, - \vdash \Delta, A \vee B, B}{\Gamma \vdash \Delta, A \vee B}\ (\vee R2)$$

$$\frac{\Gamma_1, A \vee B, \Gamma_2, A \vdash \Delta \quad \Gamma_1, A \vee B, \Gamma_2, B \vdash \Delta}{\Gamma_1, A \vee B, \Gamma_2 \vdash \Delta}\ (\vee L)$$

$$\frac{\Gamma, A \vdash \Delta, A \to B}{\Gamma \vdash \Delta, A \to B}\ (\to R1) \quad \frac{\Gamma, - \vdash \Delta, A \to B, B}{\Gamma \vdash \Delta, A \to B}\ (\to R2)$$

$$\frac{\Gamma_1, A \to B, \Gamma_2, - \vdash \Delta, A \quad \Gamma_1, A \to B, \Gamma_2, B \vdash \Delta}{\Gamma_1, A \to B, \Gamma_2 \vdash \Delta}\ (\to L)$$

$$\frac{\Gamma, - \vdash \Delta, \forall x A, A[m/x] \quad (\text{for all } m)}{\Gamma \vdash \Delta, \forall x A}\ (\forall R) \qquad \frac{\Gamma_1, \forall x A, \Gamma_2, A[m/x] \vdash \Delta}{\Gamma_1, \forall x A, \Gamma_2 \vdash \Delta}\ (\forall L)$$

$$\frac{\Gamma, - \vdash \Delta, \exists x A, A[m/x]}{\Gamma \vdash \Delta, \exists x A}\ (\exists R) \qquad \frac{\Gamma_1, \exists x A, \Gamma_2, A[m/x] \vdash \Delta \quad (\text{for all } m)}{\Gamma_1, \exists x A, \Gamma_2 \vdash \Delta}\ (\exists L)$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, - \vdash \Delta, A}\ (weak\ R) \qquad \frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\ (weak\ L)$$

Figure 2 Inference Rules of IPA$^-$

a sequent in the sequent calculus LK in the standard model of numbers as follows: If $\Gamma_0, -, \Gamma_1, \ldots, -, \Gamma_n \vdash A_1, \ldots, A_n$ is provable, then (1) $\Gamma_0 \vdash$ is true, or (2) $\Gamma_0, \Gamma_1, \ldots, \Gamma_i \vdash A_i$ is true for some i. Each inference rule is sound by this interpretation. Theorem 6.1 will provide more information.

If $\Gamma_1, -, \Pi, -, \Gamma_2 \vdash \Delta$ is provable, then $\Gamma_1, -, \Pi', -, \Gamma_2 \vdash \Delta$ is provable where Π does not contain $-$ symbols and Π' is obtained from Π by exchange, weakening, and contraction. This will be shown in Proposition 3.5. On the other hand, we cannot use right exchange, nor left exchange over formulas in different groups.

We explain this system with some examples. In the examples, we assume the identity lemma $\Gamma_1, A, \Gamma_2 \vdash \Delta, A$, which will be shown in Lemma 3.4 after the examples.

▶ Example 3.1. The first example is given in Figure 3, which shows the conjunction of IPA$^-$ is commutative.

▶ Example 3.2. The next example shows how this system respects the order of formulas. We have three provable sequents

$$-, A, -, B \vdash A, \bot,$$
$$-, A, -, B \vdash \bot, A,$$
$$-, A, -, B \vdash \bot, B.$$

$$\dfrac{\dfrac{\dfrac{}{-, A \wedge B, -, B \vdash B \wedge A, B}\,(Id)}{-, A \wedge B, - \vdash B \wedge A, B}\,(\wedge L2) \quad \dfrac{\dfrac{}{-, A \wedge B, -, A \vdash B \wedge A, A}\,(Id)}{-, A \wedge B, - \vdash B \wedge A, A}\,(\wedge L1)}{-, A \wedge B \vdash B \wedge A}\,(\wedge R)$$

Figure 3 Example 3.1

On the other hand the sequent

$$-, A, -, B \vdash B, \bot$$

is not provable. The first sequent is provable since the initial and the first groups give the assumption A, which proves the first formula A. The second sequent is provable since the initial, the first, and the second groups give the assumptions A, B, which prove the second formula A. The third sequent is provable similarly to the second sequent, since the initial, the first, and the second groups give the assumptions A, B, which prove the second formula B. Formally the first sequent is proved by

$$\dfrac{\dfrac{\dfrac{}{-, A \vdash A}\,(Id)}{-, A, - \vdash A, \bot}\,(weak\ R)}{-, A, -, B \vdash A, \bot}\,(weak\ L)$$

and the second and the third sequents are proved by (Id).

On the other hand, the fourth sequent is not provable, since we have neither of the following cases: (1) the initial group is empty, which proves the contradiction, nor (2) the initial and the first groups give the assumption A, which proves the first formula B, nor (3) the initial, the first, and the second groups give the assumptions A, B, which prove the second formula $\bot$.

▶ Example 3.3. Suppose P be a predicate symbol. Let $A(x) = \exists y P(x, y)$. The following is a proof of an instance $\forall x(A(x) \vee \neg A(x))$ of EM_1.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots\ \pi_m}{-, -, P(n,m) \vdash A(n) \vee \neg A(n), \neg A(n)}\,(\text{for all } m)}{-, -, A(n) \vdash A(n) \vee \neg A(n), \neg A(n)}}{-, - \vdash A(n) \vee \neg A(n), \neg A(n)}}{- \vdash A(n) \vee \neg A(n)}\,(\text{for all } n)}{- \vdash \forall x(A(x) \vee \neg A(x))}$$

If $P(n, m)$ is false, the proof π_m is the axiom $(Ax\ L)$. If $P(n, m)$ is true, the proof π_m is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{-, -, - \vdash A(n) \vee \neg A(n), A(n), P(n,m)}\,(Ax\ R)}{-, - \vdash A(n) \vee \neg A(n), A(n)}}{- \vdash A(n) \vee \neg A(n)}}{-, - \vdash A(n) \vee \neg A(n), \neg A(n)}\,(weak\ R)}{-, -, P(n,m) \vdash A(n) \vee \neg A(n), \neg A(n)}\,(weak\ L)$$

Remark that $A \vee \neg A$ is not provable for a Π^0_2 formula because this system does not have exchange rules.

We will explain game theoretic semantics in a general way first. Backtracking is a feature we may add to any formal game G between two players, E (Eloise) and A (Abelard), defining

$$\dfrac{\dfrac{}{\Gamma_1, B \to C, \Gamma_2, B, - \vdash \Delta, B \to C, B}\ (Id) \quad \dfrac{\dfrac{}{\Gamma_1, B \to C, \Gamma_2, B, C, - \vdash \Delta, B \to C, C}\ (Id)}{\Gamma_1, B \to C, \Gamma_2, B, C \vdash \Delta, B \to C}\ (\to R2)}{\dfrac{\Gamma_1, B \to C, \Gamma_2, B \vdash \Delta, B \to C}{\Gamma_1, B \to C, \Gamma_2 \vdash \Delta, B \to C}\ (\to R1)}\ (\to L)$$

Figure 4 Proof of Lemma 3.4

a new game bck(G). Informally, for a while a play on bck(G) runs like a play in G. However, in addition to the moves of G, Player E (Eloise) can make a new kind of move, called backtracking. We imagine that each new position of the play p is added to some stack. Using backtracking, E can move back to some previous position p of the play, provided p is recorded in the stack, erase all positions of the stack coming after p in the stack, and eventually perform an ordinary move from p (this new move is added to the stack).

In some cases, E has no recursive winning strategies over G, but some recursive winning strategies if we allow her to backtrack (i.e. E has some recursive winning strategy over bck(G)). "Backtracking" allows E to win more games using recursive winning strategies. The intuitive reason is that, in bck(G), E is not forced to provide a winning move at once, but she can find this winning move after several attempts, by trial-and-error.

Backtracking defines a new method for unwinding proofs. Assume that A is any implication-free arithmetical formula, and call TA the Tarski game for A. Then E has a recursive winning strategy over TA if and only if A is intuitionistically provable. In contrast, E has recursive winning strategy over bck(TA) if and only if A has a classical proof using only EM_1.

Assume the players play A_i at i-the stage. Then the stack of the plays is represented by $A_1, A_2, \ldots, A_n$. This stack can be simulated by the sequent $A_1, A_2, \ldots, A_n$, if the sequent calculus does not have the exchange rules and it respects the order of formulas. The weakening rules give backtracking, since it changes $A_1, A_2, \ldots, A_n, B$ to $A_1, A_2, \ldots, A_n$. Following this idea, for the positive fragment of IPA^-, [4] showed that it has 1-backtracking game semantics, and a proof in the system corresponds to a winning strategy in the game. Kobayashi [10] and the authors of this paper are preparing to show that IPA^- has a nice game theoretic semantics with 1-backtracking.

We will present several properties of IPA^-.

▶ Lemma 3.4. The following is derivable.

$$\dfrac{}{\Gamma_1, A, \Gamma_2 \vdash \Delta, A}\ (Id)$$

This lemma is shown by induction on A in a standard way. We explain only the implication case $A = B \to C$, which is proved in Figure 4.

$\sharp_{-}\Gamma$ denotes the number of the symbol $-$ in Γ. $(\Gamma)_0$ is defined to be Γ if Γ does not contain $-$. $(\Gamma, -, \Pi)_0$ is defined to be Γ if Γ does not contain $-$.

We will show some structural rules are admissible in this system. In the antecedent, we can use weakening by (*weak L*2), and contraction by (*cont L*). In the same group in the antecedent, we can also use exchange by (*exch L*).

▶ Proposition 3.5. (1) The following are derivable.

$$\dfrac{}{\Gamma_1, A, \Gamma_2 \vdash \Delta}\ (Ax\ L2)\ (A \text{ a false atomic formula})$$
$$\dfrac{}{\Gamma \vdash \Delta_1, A, \Delta_2}\ (Ax\ R2)\ (A \text{ a true atomic formula})$$

(2) The following are admissible.

$$\frac{\Gamma_1, \top, \Gamma_2 \vdash \Delta}{\Gamma_1, \Gamma_2 \vdash \Delta}\ (\top E) \qquad \frac{\Gamma_1, -, \Gamma_2, -, \Gamma_3 \vdash \Delta_1, A, A, \Delta_2}{\Gamma_1, -, \Gamma_2, \Gamma_3 \vdash \Delta_1, A, \Delta_2}\ (cont\ R)\ (\sharp_- \Gamma_3 = |\Delta_2|, \sharp_- \Gamma_2 = 0)$$

$$\frac{\Gamma_1, -, \Gamma_2 \vdash \Delta_1, \bot, \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ (\bot E)\ (\sharp_- \Gamma_2 = |\Delta_2|) \qquad \frac{\Gamma_1, \Gamma_2 \vdash \Delta}{\Gamma_1, A, \Gamma_2 \vdash \Delta}\ (weak\ L2)$$

$$\frac{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}{\Gamma_1, -, \Gamma_2 \vdash \Delta_1, A, \Delta_2}\ (weak\ R2)\ (\sharp_- \Gamma_2 = |\Delta_2|, (\Gamma_2)_0 = \phi)$$

$$\frac{\Gamma \vdash \Delta}{\Pi, \Gamma \vdash \Sigma, \Delta}\ (weak\ R3)\ (\sharp_- \Pi = |\Sigma|)$$

$$\frac{\Gamma_1, -, A, \Gamma_2 \vdash \Delta}{\Gamma_1, A, -, \Gamma_2 \vdash \Delta}\ (move) \qquad \frac{\Gamma_1, A, \Gamma_2, A, \Gamma_3 \vdash \Delta}{\Gamma_1, A, \Gamma_2, \Gamma_3 \vdash \Delta}\ (cont\ L) \qquad \frac{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Gamma_1, B, A, \Gamma_2 \vdash \Delta}\ (exch\ L)$$

4 Cut Elimination in IPA$^-$

We will show the cut elimination theorem for IPA$^-$.

▶ Definition 4.1. We define the cut rule:

$$\frac{\Gamma_1, - \vdash \Gamma_2, A \quad \Delta_1, A, \Sigma_1 \vdash \Delta_2, \Sigma_2}{\Gamma_1, \Delta_1, \Sigma_1 \vdash \Gamma_2, \Delta_2, \Sigma_2}\ (cut)$$

where $\sharp_- \Sigma_1 = |\Sigma_2|$.

We have the cut elimination theorem in IPA$^-$.

▶ Theorem 4.2 (Cut Elimination). If $\Gamma \vdash \Delta$ is provable in IPA$^-$ with the cut rule, then it is provable in IPA$^-$.

In order to prove this theorem, we use the following rule:

$$\frac{\Gamma_1, - \vdash \Gamma_2, A \quad \Gamma_1, A, \Delta_1 \vdash \Gamma_2, \Delta_2}{\Gamma_1, \Delta_1 \vdash \Gamma_2, \Delta_2}\ (cut2)$$

In the next lemma we will prove the rule $(cut2)$ can be eliminated.

▶ Lemma 4.3. (1) If we have a proof

$$\begin{array}{l} \vdots\ \pi_1 \\ \Gamma_1, -, \Pi_1 \vdash \Gamma_2, A, \Pi_2 \end{array}$$

in IPA$^-$ where $\sharp_- \Gamma_1 = |\Gamma_2|$, and the proof π_2

$$\frac{\begin{array}{c}\vdots \\ \Gamma_1, A, \Delta_1^i \vdash \Gamma_2, \Delta_2^i \quad (i \in I)\end{array}}{\Gamma_1, A, \Delta_1 \vdash \Gamma_2, \Delta_2}\ (Rule)$$

in IPA$^-$ where $(Rule)$ is a logical rule which introduces the formula A, and we have proofs

$$\begin{array}{l} \vdots\ \pi_3^i \\ \Gamma_1, \Delta_1^i \vdash \Gamma_2, \Delta_2^i \end{array}$$

for $i \in I$ in IPA$^-$, then $\Gamma_1, \Delta_1, \Pi_1 \vdash \Gamma_2, \Delta_2, \Pi_2$ is provable in IPA$^-$.

(2) If we have a proof

$$\dfrac{\begin{matrix}\vdots\ \pi_1 \\ \Gamma_1, - \vdash \Gamma_2, A\end{matrix} \quad \begin{matrix}\vdots\ \pi_2 \\ \Gamma_1, A, \Delta_1 \vdash \Gamma_2, \Delta_2\end{matrix}}{\Gamma_1, \Delta_1 \vdash \Gamma_2, \Delta_2}\ (cut2)$$

in IPA^- with the rule $(cut2)$ and the subproofs π_1 and π_2 do not contain the rule $(cut2)$, then the conclusion $\Gamma_1, \Delta_1 \vdash \Gamma_2, \Delta_2$ is provable in IPA^-.

$|\pi|$ denotes the height of the proof π. We can prove (1) and (2) simultaneously by induction on $(A, |\pi_1| + |\pi_2|)$. For each step, we will first show (1) and use (1) to show (2).

5 From IHA + EM_1 to IPA^-

This section proves the implication from IHA+EM_1 to IPA^-. We will use the cut elimination theorem for the proof.

▶ Proposition 5.1. (1) If $\Gamma \vdash \Delta$ is provable in IHA, then $-^{|\Pi|+|\Delta|}, \Gamma \vdash \Pi, \Delta$ is provable in IPA^- for any Π.

(2) If $\Gamma \vdash A$ is provable in IHA, then $-, \Gamma \vdash A$ is provable in IPA^-.

The proof idea is simulating each inference rule of IHA by inference rules of IPA^-. One difference is that a logical rule in IPA^- has a redundant principal formula. For example, the right conjunction rule in IPA^- is

$$\dfrac{\Gamma, - \vdash \Delta, A \wedge B, A \quad \Gamma, - \vdash \Delta, A \wedge B, B}{\Gamma \vdash \Delta, A \wedge B}\ (\wedge R)$$

and on the other hand the right conjunction rule in IHA is

$$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ (\wedge R)$$

This difference is covered by putting $A \wedge B$ by $(weak\ R2)$ in Proposition 3.5. The other difference is the existence of $-$, which is handled by moving $-$ by $(move)$ in Proposition 3.5.

▶ Theorem 5.2. If $\Gamma, \mathrm{EM}_1 \vdash A$ is provable in IHA, then $-, \Gamma \vdash A$ is provable in IPA^-.

Proof. By Proposition 5.1 (2), $-, \Gamma, \mathrm{EM}_1 \vdash A$ is provable in IPA^-.

We can show $- \vdash \mathrm{EM}_1$ in IPA^- in a similar way to Example 3.3. By (cut) we have $-, \Gamma \vdash A$ in IPA^- with (cut). By Theorem 4.2, we have $-, \Gamma \vdash A$ in IPA^-. □

6 From IPA^- to IHA + EM_1

This section proves the direction from IPA^- to IHA + EM_1.

In order to discuss proofs in infinitary logic, we will have to formalize proofs in codes and discuss some recursive functions from proofs to proofs. However, for space limitation, we will not describe those codes in details.

Since it is well known that the cut elimination theorem holds for IHA [12], we will use the following cut rule in IHA:

$$\dfrac{\Gamma \vdash A \quad \Pi, A \vdash \Sigma}{\Gamma, \Pi \vdash \Sigma}\ (cut)$$

In our proof, we will use the idea of flag formulas. A flag formula F_i is a Π^0_1 formula and is assigned to each formula A_i in a sequent $\Gamma \vdash A_1, \ldots, A_n$ when its proof is given. Given a

proof of $\Gamma \vdash A_1, \ldots, A_n$, if F_i is true, then the proof does not include any sequent with its succedent of length less than i.

In order to prove the theorem, we need the minimum length of the succedents in the sequents in a given proof. Even if a proof is given, we cannot effectively find the minimum length since a proof may be infinite. The idea is using a set of Π^0_1 formulas to describe the minimum length. When $F_1, \ldots, F_m$ are true and $F_{m+1}, \ldots, F_n$ are false, we know the minimum length is m. The point is that we can effectively assign these Π^0_1 formulas even for an infinite proof.

For example, for the proof

$$\dfrac{\dfrac{\dfrac{}{-, A \vdash A}\ (Ax\ R)}{-, A, - \vdash A, \bot}\ (weak\ R)}{-, A, -, B \vdash A, \bot}\ (weak\ L)$$

where A and B are true atomic formulas. Let F_1 and F_2 be the flag formulas for A and $\bot$ respectively. We will define F_1 to be true and F_2 to be false, which means the minimum length is 1.

We will explain how to define flag formulas by example. The first example is a proof ending with the rule $(\wedge R)$:

$$\dfrac{\begin{array}{cc}\vdots\ \pi_1 & \vdots\ \pi_2 \\ \Gamma, - \vdash \Delta, A \wedge B, A & \Gamma, - \vdash \Delta, A \wedge B, B\end{array}}{\Gamma \vdash \Delta, A \wedge B}\ (\wedge R)$$

Let the minimum length be m and the minimum length for π_i be m_i for $i = 1, 2$. Let the flag formulas for the proof be $F_1, \ldots, F_n$, and the flag formulas for π_i be $F^i_1, \ldots, F^i_n$ for $i = 1, 2$. We can calculate m by $m = \min(m_1, m_2)$, but we do it by using flag formulas instead. We define $F_j = F^1_j \wedge F^2_j$ for $j = 1, \ldots, n$.

The second example is a proof ending with the rule $(\forall R)$:

$$\dfrac{\begin{array}{c}\vdots\ \pi_k \\ \Gamma, - \vdash \Delta, \forall x A, A[k/x] \quad (\text{for all } k)\end{array}}{\Gamma \vdash \Delta, \forall x A}\ (\forall R)$$

Let the minimum length be m and the minimum length for π_k be m_k. Let the flag formulas for the proof be $F_1, \ldots, F_n$, and the flag formulas for π_k be $F^k_1, \ldots, F^k_n$. We cannot effectively calculate $m = \min_k(m_k)$, but we can do it by flag formulas. We define $F_j = \forall x F^x_j$ for $j = 1, \ldots, n$, which informally means the infinite conjunction $F^1_j \wedge F^2_j \wedge \ldots$. Note that F_j is also a Π^0_1 formula when F^k_j $(k = 0, 1, 2, \ldots)$ are Π^0_1.

▶ Theorem 6.1. There exists a recursive function such that if IPA^- proves the sequent $\Gamma_0, -, \Gamma_1, -, \Gamma_2, \ldots, -, \Gamma_n \vdash A_1, A_2, \ldots, A_n$ where $n \geq 1$ and Γ_i does not contain any $-$ symbol, then the function computes the codes of Π^0_1 formulas $F_1, \ldots, F_n$ and the codes of proofs of the following in $\mathrm{IHA} + \mathrm{EM}_1$ from the code of the proof of the sequent $\Gamma_0, -, \Gamma_1, -, \Gamma_2, \ldots, -, \Gamma_n \vdash A_1, A_2, \ldots, A_n$:

(1) $\neg F_1, \Gamma_0 \vdash$,
(2) $F_i, \neg F_{i+1}, \Gamma_0, \ldots, \Gamma_i \vdash A_i \quad (1 \leq i < n)$,
(3) $F_n, \Gamma_0, \ldots, \Gamma_n \vdash A_n$.

▶ Theorem 6.2. If IPA^- proves $\Gamma_0, -, \Gamma_1 \vdash A$, then $\mathrm{IHA} + \mathrm{EM}_1$ proves $\Gamma_0, \Gamma_1 \vdash A$.

Proof. By Theorem 6.1 with $n = 1$, there exists the Π^0_1 formula F_1 such that $\mathrm{IHA} + \mathrm{EM}_1$ proves (1) $\neg F_1, \Gamma_0 \vdash$, and (2) $F_1, \Gamma_0, \Gamma_1 \vdash A$. By weakening and $(\vee L)$, we get $\neg F_1 \vee F_1, \Gamma_0, \Gamma_1 \vdash A$. We have $\mathrm{EM}_1 \vdash \neg F_1 \vee F_1$. Hence, by the cut rule, we have the claim. □

$$\frac{}{\Gamma_1, A, \Gamma_2 \vdash \Delta}\ (Ax\ L)\ (A \text{ a false atomic formula})$$

$$\frac{}{\Gamma \vdash \Delta_1, A, \Delta_2}\ (Ax\ R)\ (A \text{ a true atomic formula})$$

$$\frac{\Gamma, \top \vdash \Delta, A \wedge B, A \quad \Gamma, \top \vdash \Delta, A \wedge B, B}{\Gamma \vdash \Delta, A \wedge B}\ (\wedge R)$$

$$\frac{\Gamma_1, A \wedge B, \Gamma_2, A \vdash \Delta, D, D}{\Gamma_1, A \wedge B, \Gamma_2 \vdash \Delta, D}\ (\wedge L1) \qquad \frac{\Gamma_1, A \wedge B, \Gamma_2, B \vdash \Delta, D, D}{\Gamma_1, A \wedge B, \Gamma_2 \vdash \Delta, D}\ (\wedge L2)$$

$$\frac{\Gamma, \top \vdash \Delta, A \vee B, A}{\Gamma \vdash \Delta, A \vee B}\ (\vee R1) \qquad \frac{\Gamma, \top \vdash \Delta, A \vee B, B}{\Gamma \vdash \Delta, A \vee B}\ (\vee R2)$$

$$\frac{\Gamma_1, A \vee B, \Gamma_2, A \vdash \Delta, D, D \quad \Gamma_1, A \vee B, \Gamma_2, B \vdash \Delta, D, D}{\Gamma_1, A \vee B, \Gamma_2 \vdash \Delta, D}\ (\vee L)$$

$$\frac{\Gamma, A \vdash \Delta, A \to B, A \to B}{\Gamma \vdash \Delta, A \to B}\ (\to R1) \qquad \frac{\Gamma, \top \vdash \Delta, A \to B, B}{\Gamma \vdash \Delta, A \to B}\ (\to R2)$$

$$\frac{\Gamma_1, A \to B, \Gamma_2, \top \vdash \Delta, D, A \quad \Gamma_1, A \to B, \Gamma_2, B \vdash \Delta, D, D}{\Gamma_1, A \to B, \Gamma_2 \vdash \Delta, D}\ (\to L)$$

$$\frac{\Gamma, \top \vdash \Delta, \forall x A, A[m/x] \quad (\forall m)}{\Gamma \vdash \Delta, \forall x A}\ (\forall R) \qquad \frac{\Gamma_1, \forall x A, \Gamma_2, A[m/x] \vdash \Delta, D, D}{\Gamma_1, \forall x A, \Gamma_2 \vdash \Delta, D}\ (\forall L)$$

$$\frac{\Gamma, \top \vdash \Delta, \exists x A, A[m/x]}{\Gamma \vdash \Delta, \exists x A}\ (\exists R) \qquad \frac{\Gamma_1, \exists x A, \Gamma_2, A[m/x] \vdash \Delta, D, D \quad (\forall m)}{\Gamma_1, \exists x A, \Gamma_2 \vdash \Delta, D}\ (\exists L)$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta, B}\ (sweak) \qquad \frac{\top, \Gamma \vdash \Delta}{\Gamma \vdash \Delta}\ (\top E) \qquad \frac{\Gamma \vdash \bot, \Delta}{\Gamma \vdash \Delta}\ (\bot E)$$

Figure 5 Inference Rules of NCIPA

7 Non-Commutative Infinitary Peano Arithmetic

We define non-commutative infinitary Peano arithmetic NCIPA. In the next section we will prove that NCIPA is a subsystem of IPA essentially equivalent to IPA$^-$.

The language is defined to be the same as that of IPA. The inference rules are given by Figure 5. The rule $(sweak)$ means symmetric weakening. A proof in this system is defined as a well-founded recursive tree in a similar way to IPA.

Intuitive meaning of provable sequents is given by using the familiar interpretation of a sequent in the sequent calculus LK in the standard model of numbers as follows: If $\Pi, A_1, \ldots, A_n \vdash B_1, \ldots, B_n$ is provable, then (1) $\Pi \vdash$ is true, or (2) $\Pi, A_1, \ldots, A_i \vdash B_i$ is true for some i. If $A_1, \ldots, A_n \vdash C_1, \ldots, C_m, B_1, \ldots, B_n$ is provable, then (1) $\vdash C_i$ is true for some i, or (2) $A_1, \ldots, A_i \vdash B_i$ is true for some i.

This system is obtained from IPA$^-$ by coding grouping information by the length of a sequence of formulas. We explain it by example.

▶ Example 7.1. The sequent

$$A_1, -, A_2, A_3, -, A_4, -, A_5, A_6 \vdash B_1, B_2, B_3$$

$$\dfrac{\dfrac{\dfrac{}{A\wedge B,\top,B\vdash B\wedge A,B,B}\,(Id)}{A\wedge B,\top\vdash B\wedge A,B}\,(\wedge L2) \quad \dfrac{\dfrac{}{A\wedge B,\top,A\vdash B\wedge A,A,A}\,(Id)}{A\wedge B,\top\vdash B\wedge A,A}\,(\wedge L1)}{A\wedge B\vdash B\wedge A}\,(\wedge R)$$

Figure 6 Example 7.2

in IPA$^-$ is coded by the sequent

$$A_1,\top,A_2,A_3,\top,A_4,\top,A_5,A_6\vdash B_1,B_1,B_1,B_2,B_2,B_3,B_3,B_3$$

in NCIPA. The atomic formula $\top$ is used for separating groups. The group $\top,A_5,A_6$ corresponds to B_3,B_3,B_3. The group $\top,A_4$ corresponds to B_2,B_2. The group $\top,A_2,A_3$ corresponds to B_1,B_1,B_1. We can decode this information by counting formulas from the right to the left on both sides. This decoding may not be unique, but it is unique up to the provability in IPA$^-$. This translation is formally defined in Definition 8.4.

We explain this system by the same examples as those in Section 3. In the examples, we assume the identity lemma $\Gamma_1,A,\Gamma_2\vdash\Delta,A$, which will be shown as Lemma 7.4 after the examples.

▶ Example 7.2. The first example in Figure 6 shows the conjunction of NCIPA is commutative.

▶ Example 7.3. The next example shows how this system respects the order of formulas. We have three provable sequents

$A,B\vdash A,\bot,$
$A,B\vdash\bot,A,$
$A,B\vdash\bot,B.$

On the other hand the sequent

$$A,B\vdash B,\bot$$

is not provable. The first sequent is provable since $A\vdash A$ is true. The second sequent is provable since $A,B\vdash A$ is true. The third sequent is provable since $A,B\vdash B$ is true. Formally the first sequent is proved by

$$\dfrac{\dfrac{}{A\vdash A}\,(Id)}{A,B\vdash A,\bot}\,(sweak)$$

and the second and the third sequents are proved by (Id). On the other hand, the fourth sequent is not provable, since $A\vdash B$ is not true and $A,B\vdash\bot$ is not true.

▶ Lemma 7.4. The following is derivable.

$$\dfrac{}{\Gamma_1,A,\Gamma_2\vdash\Delta,A}\,(Id)$$

This lemma is shown by induction on A in a similar way to Lemma 3.4.

Remark. (1) $(\bot E)$ is necessary for making a binary left logical rule for the empty succedent admissible. It is used in the proof of Proposition 8.6. For example, the following is admissible.

$$\dfrac{\Gamma_1,A\vee B,\Gamma_2,A\vdash\quad\Gamma_1,A\vee B,\Gamma_2,B\vdash}{\Gamma_1,A\vee B,\Gamma_2\vdash}\,(\vee L)$$

(2) $(\top E)$ is necessary since $\vdash \top \vee \bot, \bot$ would not be provable otherwise, though it is indeed provable by

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\top \vdash \top \vee \bot, \top}\,(Ax\ R)}{\vdash \top \vee \bot}\,(\vee R1)}{\top \vdash \top \vee \bot, \bot}\,(sweak)}{\vdash \top \vee \bot, \bot}\,(\top E)$$

▶ Proposition 7.5. The following are admissible.

$$\dfrac{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}{\Gamma_1, A, \Gamma_2 \vdash \Delta_1, B, \Delta_2}\,(sweak2) \quad (|\Gamma_2| = |\Delta_2|)$$

$$\dfrac{\Gamma_1, A, A, \Gamma_2 \vdash \Delta_1, B, B, \Delta_2}{\Gamma_1, A, \Gamma_2 \vdash \Delta_1, B, \Delta_2}\,(scont) \quad (|\Gamma_2| = |\Delta_2|)$$

$$\dfrac{\Gamma_1, \Gamma_2 \vdash \Delta}{\Gamma_1, A, \Gamma_2 \vdash \Delta}\,(weak\ L) \qquad \dfrac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}\,(\bot I) \qquad \dfrac{\Gamma_1, \top, \Gamma_2 \vdash \Delta}{\Gamma_1, A, \Gamma_2 \vdash \Delta}\,(replace\ L)$$

$$\dfrac{\Gamma \vdash \Delta_1, A, A, \Delta_2}{\Gamma \vdash \Delta_1, A, \Delta_2}\,(cont\ R) \qquad \dfrac{\Gamma_1, \top, A, \Gamma_2 \vdash \Delta_1, B, B, \Delta_2}{\Gamma_1, A, \Gamma_2 \vdash \Delta_1, B, \Delta_2}\,(\top E2) \quad (|\Gamma_2| = |\Delta_2|)$$

We define the system NCIPA+EX as the system NCIPA with the following inference rules $(exch\ L)$ and $(exch\ R)$.

$$\dfrac{\Gamma \vdash \Delta_1, A, B, \Delta_2}{\Gamma \vdash \Delta_1, B, A, \Delta_2}\,(exch\ R) \qquad \dfrac{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Gamma_1, B, A, \Gamma_2 \vdash \Delta}\,(exch\ L)$$

When the exchange rules are added to NCIPA, the coding information is lost and it becomes equivalent to IPA.

▶ Theorem 7.6. $\Gamma \vdash \Delta$ is provable in NCIPA+EX if and only if $\Gamma \vdash \Delta$ is provable in IPA.

The system NCIPA is a subclassical logic.

▶ Theorem 7.7. $\Gamma \vdash A$ is provable in NCIPA if and only if $\Gamma, \mathrm{EM}_1 \vdash A$ is provable in IHA.

We will complete the proof of this theorem in Section 8.

8 Translations between NCIPA and IPA$^-$

This section gives translations between NCIPA and IPA$^-$ in both directions and proves that they preserve provability. By using these translations, we will prove the equivalence theorem between NCIPA and IHA $+ \mathrm{EM}_1$.

First, we give a translation from NCIPA to IPA$^-$. To translate $\Gamma \vdash \Delta$, we insert the same number of $-$ symbols as $|\Delta|$ into Γ by adding a single $-$ symbol in front of each formula from the right. For example, the sequent $A_1, A_2, A_3, A_4 \vdash B_1, B_2$ in NCIPA is translated into the sequent $A_1, A_2, -, A_3, -, A_4 \vdash B_1, B_2$ in IPA$^-$.

▶ Definition 8.1 (Translation from NCIPA to IPA$^-$). We translate a sequent $\Gamma \vdash \Delta$ in NCIPA into the sequent $\Gamma^{-|\Delta|} \vdash \Delta$ in IPA$^-$, where $(\Gamma_0, A_1, A_2, \ldots, A_n)^{-n}$ is defined as $\Gamma_0, -, A_1, -, A_2, \ldots, -, A_n$ and $(A_1, A_2, \ldots, A_m)^{-n}$ $(m < n)$ is defined as $-^{n-m}, -, A_1, -, A_2, \ldots, -, A_m$.

▶ Example 8.2. The NCIPA-proof of $A \wedge B \vdash B \wedge A$ in Example 7.2 is translated into the IPA$^-$-proof given in Figure 7.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{-, A\wedge B, -, \top, -, B \vdash B\wedge A, B, B}\,(Id)}{-, A\wedge B, -, \top, B \vdash B\wedge A, B}\,(cont\ R)}{-, A\wedge B, -, \top \vdash B\wedge A, B}\,(\wedge L2)}{-, A\wedge B, - \vdash B\wedge A, B}\,(\top E) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{-, A\wedge B, -, \top, -, A \vdash B\wedge A, A, A}\,(Id)}{-, A\wedge B, -, \top, A \vdash B\wedge A, A}\,(cont\ R)}{-, A\wedge B, -, \top \vdash B\wedge A, A}\,(\wedge L1)}{-, A\wedge B, - \vdash B\wedge A, A}\,(\top E)}{-, A\wedge B \vdash B\wedge A}\,(\wedge R)$$

Figure 7 Example for NCIPA to IPA$^-$

$$\dfrac{\dfrac{\dfrac{}{\top, A\wedge B, \top, B \vdash B\wedge A, B\wedge A, B, B}\,(Id)}{\top, A\wedge B, \top \vdash B\wedge A, B\wedge A, B}\,(\wedge L2) \qquad \dfrac{\dfrac{}{\top, A\wedge B, \top, A \vdash B\wedge A, B\wedge A, A, A}\,(Id)}{\top, A\wedge B, \top \vdash B\wedge A, B\wedge A, A}\,(\wedge L1)}{\top, A\wedge B \vdash B\wedge A, B\wedge A}\,(\wedge R)$$

Figure 8 Example for IPA$^-$ to NCIPA

▶ Proposition 8.3. $\Gamma \vdash_{\mathrm{NCIPA}} \Delta$ implies $\Gamma^{-|\Delta|} \vdash_{\mathrm{IPA}^-} \Delta$.

This is proved by induction on the proof.

Next, we define a translation from IPA$^-$ to NCIPA. To translate $\Gamma \vdash \Delta$, we replace $-$ by $\top$ in Γ, and the succedent is produced from Δ by multiplying the i-th formula by $n_i + 1$ when the i-th group in Γ has n_i formulas. An example is given in Example 7.1. A^n denotes $A, \ldots, A$ (n times).

▶ Definition 8.4 (Translation from IPA$^-$ to NCIPA). We translate a sequent $\Gamma \vdash \Delta$ in IPA$^-$ into the sequent $\Gamma^\top \vdash \Delta^\Gamma$ in NCIPA, where $\Gamma^\top$ is defined as $\Gamma_0, \top, \Gamma_1, \top, \Gamma_2, \ldots, \top, \Gamma_n$ and $(A_1, \ldots, A_n)^\Gamma$ is defined as $A_1^{|\Gamma_1|+1}, A_2^{|\Gamma_2|+1}, \ldots, A_n^{|\Gamma_n|+1}$ if Γ is $\Gamma_0, -, \Gamma_1, -, \Gamma_2, \ldots, -, \Gamma_n$ and Γ_i does not contain $-$.

▶ Example 8.5. The IPA$^-$-proof of $-, A\wedge B \vdash B\wedge A$ in Example 3.1 is translated into the NCIPA-proof given in Figure 8.

▶ Proposition 8.6. $\Gamma \vdash_{\mathrm{IPA}^-} \Delta$ implies $\Gamma^\top \vdash_{\mathrm{NCIPA}} \Delta^\Gamma$.

This is proved by induction on the proof.

Proof of Theorem 7.7. From the left-hand side to the right-hand side.

By Proposition 8.3, we have $\Gamma^{-1} \vdash_{\mathrm{IPA}^-} A$. By Theorem 6.2, we have $\Gamma, \mathrm{EM}_1 \vdash_{\mathrm{IHA}} A$.

From the right-hand side to the left-hand side.

By Theorem 5.2, we have $-, \Gamma \vdash_{\mathrm{IPA}^-} A$. By Proposition 8.6, we get $\top, \Gamma \vdash_{\mathrm{NCIPA}} A^{|\Gamma|+1}$. By $(\top E)$ and $(cont\ R)$ in Proposition 7.5, $\Gamma \vdash_{\mathrm{NCIPA}} A$. □

9 Cut Elimination for NCIPA

In this section, we will prove the cut elimination theorem for NCIPA.

▶ Definition 9.1. We give the cut rule in the system NCIPA.

$$\dfrac{\Gamma_1, \top \vdash \Gamma_2, A \quad \Delta_1, A, \Sigma_1 \vdash \Delta_2, \bot, \Sigma_2}{\Gamma_1, \Delta_1, \Sigma_1 \vdash \Gamma_2, \Delta_2, \Sigma_2}\,(cut)$$

where $|\Pi_1| = |\Pi_2|, |\Delta_1| = |\Delta_2|, |\Sigma_1| = |\Sigma_2|$.

We can eliminate the cut rule in NCIPA.

▶ Theorem 9.2 (Cut Elimination). If $\Gamma \vdash \Delta$ is provable in NCIPA with the rule (cut), then $\Gamma \vdash \Delta$ is provable in NCIPA.

This theorem is proved by using the cut elimination theorem for IPA$^-$ and the next proposition.

▶ Proposition 9.3 (NCIPA to IPA$^-$ to NCIPA). $(\Gamma^{-|\Delta|})^\top \vdash_{\mathrm{NCIPA}} \Delta^{\Gamma^{-|\Delta|}}$ iff $\Gamma \vdash_{\mathrm{NCIPA}} \Delta$.

10 Concluding Remarks

We showed that by removing the exchange rules, Peano arithmetic with the ω-rules becomes Heyting arithmetic with the recursive ω-rules and the Σ^0_1 excluded middle. The equivalence is an open question when the system is Peano arithmetic without the ω-rules.

Future work would be to investigate the computational content of the subclassical systems IPA$^-$ and NCIPA.

References

1 Y. Akama, S. Berardi, S. Hayashi, and U. Kohlenbach, An Arithmetical Hierarchy of the Law of Excluded Middle and Related Principles, In: *Proc. LICS2004* (2004) 192–201.

2 F. Aschieri, Interactive learning based realizability and 1-backtracking games, In: Proceedings of Third International Workshop on Classical Logic and Computation (CLC2010), *Electronic Proceedings in Theoretical Computer Science* **47** (2011) 6–20.

3 S. Berardi, Some intuitionistic equivalents of classical principles for degree 2 formulas, *Annals of Pure and Applied Logic* **139** (2006) 185–200.

4 S. Berardi and Y. Yamagata, A sequent calculus for Limit Computable Mathematics, *Annals of Pure and Applied Logic* **153** (1-3) (2008) 111–126.

5 S. Berardi, Semantics for Intuitionistic Arithmetic based on Tarski Games with retractable moves, In: Proceedings of TLCA 2007, *LNCS* **4583** (2007) 23–38.

6 S. Berardi and M. Tatsuta, Positive Arithmetic without Exchange is a Subclassical Logic, In: Proceedings of the Fifth Asian Symposium on Programming Languages and Systems (APLAS 2007), *LNCS* **4807** (2007) 271–285.

7 S. Berardi and U. de'Liguoro, A Calculus of Realizers for EM1 Arithmetic, In: Proceedings of CSL2008, *LNCS* **5213** (2008) 215–229.

8 T. Coquand, A Semantics of Evidence for Classical Arithmetic, *JSL* **60** (1) (1995) 325–337.

9 J.Y. Girard, Linear logic, *Theoretical Computer Science* **50** (1) (1987) 1–102.

10 S. Kobayashi, Game semantics for limit computable mathematics, Second NII Type Theory Workshop, 2007.

11 U. Kohlenbach, Relative Constructivity, *Journal of Symbolic Logic* **63** (4) (1998) 1218–1238.

12 P. Martin-Löf, Infinite terms and a system of natural deduction, *Compositio Mathematica* **24** (1972) 93–103.

13 M. Tatsuta, Non-Commutative First-Order Sequent Calculus, In: Proceedings of CSL2009, *LNCS* **5771** (2009) 470–484.

Model Theory in Computer Science: My Own Recurrent Themes

Johann A. Makowsky

Technion–Israel Institute of Technology
Haifa, Israel
janos@cs.technion.ac.il

Abstract

I review my own experiences in research and the management of science.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Model theory, finite model theory, databases, graph invariants

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.553

Forty Years Ago

It was exactly forty years ago, that I got my ETH-Diploma in Mathematics and Physics, followed two years later by my PhD (Dr.sc.math.). Since then I have held appointments in Zurich, Warsaw, Stanford, Vancouver, Florence, Berlin and Haifa. Scientifically, I have travelled from Model Theory proper to applications thereof in Computer Science, and finally in Combinatorics, visiting the lands of Database Theory, Specification, Verification, Artificial Intelligence, Complexity and Algorithms. I was a founding member of EACSL, its vice-president, and finally its president till 2010. But at heart I remained a mathematician with a strong interest in computer science and its foundations.

In this *retiring president's address* I would like to sketch some of the recurrent ideas of my research, and some of the lessons I have learned in managing a scientific career, and managing science as an enterprise. I concentrate here on scientific details and leave more personal remarks for the lecture. Some other personal recollections can be found in [62].

Model Theory: Categoricity and Finite Axiomatizability

My first attempt to tackle open problems was a consequence of reading M. Morley's fundamental paper on categoricity in power, [72] in the undergraduate seminar in mathematical logic at ETH Zurich, held by E. Specker and H. Läuchli, and regularly attended by the still very lucid octogenarian P. Bernays.

A first-order theory T is categorical in some infinite cardinal κ if T has no finite models and all its models of size κ are isomorphic. Morley asks, whether there is a finitely axiomatizable first-order theory T which is κ-categorical for all κ, or for all uncountable κ. Attacking these questions required understanding of the structure theory of κ-categorical theories (stable theories, rank, degree, etc.) and some idea on how to prove or disprove finite axiomatizability. I made a thorough manual literature search in the library (no `scholar.google.com` was available then) about finite axiomatizability, from which I learned about Ehrenfeucht-Fraïssé games and ultraproducts, and other methods, but only the Ehrenfeucht-Fraïssé games seemed promising to me. With some ideas on how to approach Morley's question, I attended my first logic conference in 1970, where I received encouragement and a still unpublished preprint of

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 553–567

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

[3] from A. Lachlan. Upon my return I asked my supervisor, H. Läuchli, whether I could write my M.Sc. thesis about Morley's question, and he agreed. I managed to prove

▶ **Theorem 1** ([53]).
(i) A first-order theory T which is $\aleph_0$-categorical and strongly minimal (hence categorical in all infinite kappa) cannot be finitely axiomatizable.
(ii) There is a finitely axiomatizable complete first-order theory T which is superstable.

The second edition of [10] credits me[1] with partially solving two of Morley's problems. However, most of the problems listed in [72] were solved by S. Shelah, just the finite axiomatizability questions withstood his attacks. I soon realized that this was all I could prove using the available tools. Besides Ehrenfeucht-Fraïssé games I used non-periodic tilings of the plane, an idea which was suggested to me by my supervisor H. Läuchli.

The finite axiomatizability questions were finally solved by by M. Peretyatkin [76] (there is a finitely axiomatizable ω_1-categorical theory), and by G. Cherlin, L. Harrington and A. Lachlan and by B. Zilber (there is no finitely axiomatizable theory categorical in all infinite powers), cf. [11], and [83]. G. Cherlin, L. Harrington and A. Lachlan use the classification theory of finite groups, and B. Zilber uses results on diophantine equations to overcome the difficulties I had not been able to overcome. Neither of these tools were available when I left the problem.

I presented my results on finite axiomatizability at the Logic Colloquium in Cambridge in 1971. There I met for the first time with S. Shelah, with whom I had corresponded before, and with W. Marek, who told me about student exchange programs between Switzerland and Poland. Both encounters had a major impact on my further scientific development.

▶ Lesson 1. Search and read the literature, even if it goes far back.

▶ Lesson 2. Go to conferences already as a student, but be properly prepared.

Generalized Quantifiers

It was Wiktor Marek, who introduced me in 1971 to Lindström's Theorem. It had been rediscovered by Harvey Friedman, who gave it much publicity. What a great Theorem: Predicate Logic can be characterized, among *all the logics* as the only one which satisfies the Löwenheim-Skolem Property and one of the following: compactness or axiomatizability. Well, that's the way you might promote it, but then there are plenty of details, which make it less spectacular. And still, a new paradigm was found, which consisted in characterizing logics. I immediately studied Lindström's papers and all that was known about extensions of first-order logic and prepared a seminar talk about it. Later P. Lindström told me that his original motivation for the theorem had been to find a new application of Ehrenfeucht-Fraïssé games. I spent 1972 partially in Warsaw as an exchange student, cf. [62] working under the late A. Mostowski on generalized quantifiers.

There were two lines of studying extensions of first-order logic: (i) via generalized quantifiers, and (ii) via fragments of infinitary logics. What I tried to do was to find characterizations of logics using other properties than the Löwenheim-Skolem-Tarski Theorem and the Compactness Theorem. J. Barwise showed that the admissible fragments of $\mathcal{L}_{\omega_1,\omega}$ satisfy the Craig Interpolation Theorem. D. Scott and, independently before, E. Engeler,

[1] At the time joint papers were not common practice, and H. Läuchli let me publish the results under my name alone. In [10] credit is not extended to H. Läuchli, although I clearly stated his rôle in the results.

[21, 22, 78] had shown that all countable structures over a countable vocabulary can be characterized up to isomorphism by a sentence in $\mathcal{L}_{\omega_1,\omega}$. G. Kreisel suggested his own criteria of choosing logics, [44]. In Fall 1972, E. Engeler and P. Bernays introduced me to G. Kreisel. I told him about my ideas, and he got very interested and encouraging. An intensive correspondence followed which lasted until he invited me in Fall 1973 to come to Stanford. He also provided me with a preprint of L. Tharp, published later as [81], without telling me that he was refereeing it. I naively used the material in my PhD thesis, trusting that it was given to me in good faith for use. I generalized Tharp's definitions and proved innocently theorems which may have been also on Tharp's mind.

▶ Lesson 3. Do not circulate papers you are supposed to treat confidentially.

Having worked on categorical first-order theories, I formulated and finally proved the following:

▶ **Theorem 2** ([52]). *Let $\mathcal{L}$ be a logic such that Craig's Interpolation Theorem holds for $\mathcal{L}$ and such that all countable structures over a countable vocabulary can be characterized up to isomorphism by a sentence in $\mathcal{L}$. Then $\mathcal{L}_{\omega_1,\omega}$ is a sublogic of $\mathcal{L}$.*

For this theorem I was inspiried by three papers by S. Feferman, [23, 24, 25], which among other things discuss abstract versions of the Feferman-Vaught Theorem, [26] which entered my toolbox already then.

In [52] I also announced characterization of the minimal fragments of $\mathcal{L}_{\omega_1,\omega}$ which satisfy the Sousline-Kleene version of the interpolation theorem and characterize one countable structure up to isomorphism. The same characterization was also announced by H. Friedman in the Notices of AMS. My proof used tools I had not yet properly mastered by then. However, H. Friedman's announcement brushed away my gut feelings, and made me believe that I had used the tools correctly. Only when I lectured in the Stanford Logic Seminar in fall 1973, J. Stavi showed me a counterexample to the theorem as stated in [52], and in Friedman's abstract. Together we fixed the theorem, which led to [67] and my prolonged collaboration with S. Shelah in abstract model theory.

▶ Lesson 4. Do not trust your own handwaving. Do trust your gut feelings when something is wrong with your proof.

Until 1984 much of my published work remained in abstract model theory. Abstract model theory had generated quite a bit of excitement in the logic community. This is witnessed in [5], where I contributed two chapters and co-authored one more, [57, 64, 56]. But after the publication of [5] the philosophically minded, including P. Lindström himself, lost interest, because, among other reasons, my theorems with S. Shelah had introduced large cardinals into the field, spoiling the hope for neat theorems.

S. Shelah taught me:

▶ Lesson 5. Never let aesthetics or ideology prevent you from proving a theorem!

But the reality of the mathematical community knows a law of diminishing return:

▶ Lesson 6. Not everybody who asks a mathematical question is willing to hear the answer if it requires too much time, energy or skill to understand it.

J. Stavi and S. Shelah invited me to Israel to work with them. Later romantic involvement strengthened the Israeli connection. Finally, I founded a family and stayed. But to find a job in Israel, I had to move to applications of logic in computer science.

The Promised Land: Computer Science Logic

I was well aware that mathematical logic, especially model theory, had something to offer to theoretical computer science. I had attended the Specker-Strassen Seminar in Zurich in the early 70s, cf. [80], where we studied evolving complexity theory. It was finally E. Shamir in Jerusalem who gave me the crucial impulse to approach computer science successfully. In 1978 he arranged for a "blind date" with C. Beeri, who was struggling to find the right definition of database dependencies. He also told me to attend the ACM-STOC conference in 1979 in Atlanta, where I got acquainted with V. Pratt and his dynamic logic.

▶ Lesson 7. When you change fields, do not leave your old toolbox behind!

I tried to identify problems in theoretical computer science which could be tackled using model theoretic methods. I was looking for model theoretic characterizations of certain classes of syntactically defined formulas and for analogues of Lindström's Theorems. At the Logic Colloquium 1982 in Florence I was an invited speaker and I gave a talk on *Model theoretic issues in theoretical computer science: Relational Data Bases and Abstract Data Types*, [55]. Y. Gurevich discussed this paper with me at great length in the years 1982-84, and it inspired him to write his [34]. But I had written my paper for the wrong audience: The Logicians were not interested in Computer Science, and the first LiCS conference was held only in 1986. The first CSL conference was held in 1987, and EACSL was founded in 1992.

▶ Lesson 8. One can be too early and in the wrong place at the same time.

The Fundamental Problem of Databases

My first published paper in theoretical computer science was an application of abstract model theory to dynamic logic, [48]. But my truly first result was in databases. In 1978 C. Beeri spent many hours trying to explain to me what J. Ullman had declared to be the *Fundamental Problem of Databases*. Imprecisely stated, it was the decision problem for database dependencies which at the time were meant to be generalizations of Functional Dependencies. Finally, C. Beeri accepted my suggestion, that database dependencies are to be identified with a certain subclass of universal-existential Horn formulas in purely relational first-order logic where satisfiability is restricted to finite relational structures. Actually, we defined the four classes of dependencies which later became known as FID (full implicational) and EID (embedded implicational) with the subclasses of equality generating and tuple generating dependencies. I showed C. Beeri, that in the case of EID's, the decision problem was undecidable, and suspected it to be well known. A quick consultation with M. Rabin confirmed my suspicion, although, as it turned out, M. Rabin was not quite right. My reduction used the word problem of finite semigroups. Rabin thought that this was known to be undecidable, and undecidable it was, but not well known. It had been proven by Y. Gurevich in 1966 and published only in Russian as [33]. Rabin did not know of it and confused it with some other well known decision problem. As a result of Rabin's remarks I turned my back to databases and looked for other topics. I did not realize then that, even if I could solve the technical problem, I still did not understand why solving it was important for databases. C. Beeri continued to work on this with his PhD student M. Vardi, and much of what we discussed together was further elaborated in Vardi's thesis. I tried my luck, without success, in improving Galil's lower bound for the worst case run time of the Davis-Putnam procedure, [27]. I also tried my luck, again without success, in understanding the complexity of computing the permanent. I returned to both of these topics much later.

In spring 1980 I was guest of V. Pratt at MIT. Visiting Princeton, I attended a colloquium lecture where Ullman's *Fundamental Problem of Databases* was mentioned again as the most important open problem in database theory. I told the speaker that I had solved it a year before, and he mentioned rumors that A. Chandra and H. Lewis also just solved it. Back at MIT, I showed C. Papadimitriou my proof and asked him about the rumor that A. Chandra and H. Lewis had obtained the same result. He confirmed and was kind enough to arrange that this coincidence would result in two joint papers, [9, 8].

▶ **Theorem 3** (A. Chandra, H. Lewis and JAM, [9, 8])**.**

(i) *The decision problem for* embedded implicational dependencies *is undecidable.*

(ii) *The decision problem for* full implicational dependencies *decidable and complete in exponential time, even for the typed case.*

The distinction between typed and untyped dependencies seemed to me cosmetic but was considered important to the database community. Typed here means that we look at many-sorted finite structures where sorts correspond to attributes. The undecidability of *typed* EID remained in our paper open. After that I tried to learn the true problems of database theory. However, J. Ullman changed his mind and declared that Dependency Theory and Design Theory had run their course. As a result, papers dealing with these topics were almost banned from the relevant conferences.

▶ Lesson 9. Do not get discouraged when you are told without proper references that your result is well known.

▶ Lesson 10. Not every problem which looks easy from where you stand is easy for others approaching the problem from a different angle.

▶ Lesson 11. The fact that you can solve technical problems in other people's domain, does not make you an expert in this domain.

Program Correctness and Termination

I have three papers dealing somehow with logic and program termination. One is an application of abstract model theory to dynamic logic, [48], one is a completeness theorem for a proof rule for fair termination, [32], and one contains weak second-order characterizations of various program verification systems, [51]. [32] is a good example of the previous lesson: I was able to provide a proof of a theorem formulated by N. Francez and O. Grumberg on fair termination, see [32], without grasping the essence of the problem. I really understood the problem only after reading the paper by D. Lehmann, A. Pnueli and J. Stavi, [45], which presented a different approach to fair termination.

More significantly, there was also a cultural problem: Discussing the problem with J. Stavi, he suggested that all this was a trivial application of J. Shoenfield's Tree Lemma, [79]. We went through this idea together and indeed came to the conclusion, that technically there was not much new, and that unwinding trees in special cases would be a good topic for PhD, or rather MSc students. We both grossly underestimated the gap between a logician who had studied all of Shoenfield's book [79], and a computer scientist who was interested in a particular application. The gap is not only technical, but also on the levels of abstraction. I tried to find graduate students to explore uses of the Tree Lemma, I tried also to collaborate with my colleagues, but I failed to bridge this gap. They could not believe that such a general lemma would help them, and they were not willing to spend the time to learn what appeared to them exotically abstract. One day, in 1981, I told D. Harel about my failure in recruiting partners for this project. I told him about my discussions with J. Stavi, and I

sketched to him to use of the Tree Lemma. But David was busy with other research projects. Nevertheless, somehow our discussion made him think about all this, which led to [35]. He finally published a beautiful journal paper, [36] in which he acknowledges our suggestion.

▶ Lesson 12. Do not underestimate the amount of work and ingenuity sometimes needed in applying a clean abstract theorem to a concrete problem.

▶ Lesson 13. It turns out that it is sometimes easier to reinvent the wheel for special applications.

More recently, while applying logic to graph polynomials, the Tree Lemma appeared again: first in my work with B. Godlin and E. Katz, [28], and then, while trying to turn the abstract result of [28] into a concrete result, in my work with I. Averbouch and B. Godlin to [2].

The Fundamental Problem of Database Design

The big problem of database design is the *choice of the basic relations* and the development of a *restructuring technology*. It does not matter whether we are in the Entity-Relationship model or the Relational Model of Database. There are many attempts to formulate criteria for a good choice of basic relations, some of them heuristical, some of them with a solid body of techniques, theorems and algorithms. Normal form theory is widely taught and popular because it lends itself readily to exam problems. But the last word in the design of databases has not been said. Many databases were designed fifty years ago, have become old-fashioned and have to be converted into new designs without loss of information while preserving the underlying constraints. I had three excellent students in Databases, V. Markowitz (PhD), U. Rotics (MSc), and E. Ravve (PhD). My most quoted paper in databases, [70], needed six years to get published, because the then editor of the IEEE Transactions of Software Engineering lost the paper. Only when he was replaced, the new editor hastened to publish it without sending us a referee report. I had never attended a database conference until I was an invited speaker at the ER-conference in 1996. V. Markowitz was better known than me, and at this conference many greeted me with "Ah, you were the supervisor of V. Markowitz". I naively thought that the impact of one's work was a function solely of one's results. This may still be true in the very long run, and the way my work is quoted in monographs may attest to this, but it is definitely wrong in the short run.

▶ Lesson 14. Unfortunately, you have to promote yourself by personally reporting about your work. People only read results of which they have heard before.

Horn Formulas

Horn formulas are well-known in Model Theory because they are preserved under various product constructions, cf. [10]. Product constructions are important for the algebraists (universal and other) but rarely occur in Computer Science. I encountered Horn formulas in Computer Science (not under this name) first in my discussions with C. Beeri about database dependencies. I also encountered them in the algebraic specification of data types, [47], a then very promising field of research which did not bring the results its proponents hoped for, [29, 30, 20]. And then Horn formulas started to play a central rôle in Logic Programming and rule-based reasoning. Trained as a model theorist, I started to ask myself *why Horn formulas matter in Computer Science*. In Spring 1982 I taught a course where we discussed the satisfiability problem for Horn clauses. I showed to my students that it was solvable in

polynomial (cubic) time and asked the students to come up with an algorithm which runs in less than cubic time. Oded Goldreich, then a graduate student, gave an $O(n \lg n)$-algorithm. This inspired Alon Itai and me to design a linear time algorithm. We did not think then that it this was such an important or particularly difficult result, so we did not rush into publication. A preprint was circulated in May 1982, [38], containing the result, but the main thrust of the paper was in proposing and analyzing a complexity measure for logic programming based on unit resolution and unification steps.

▶ **Theorem 4** (A. Itai and JAM, [38])**.**
Propositional HornSat is solvable in linear time.

The paper was finally published only in 1987, [39], because the referees disagreed and gave contradictory recommendations: one wanted more details in the motivation and background material, while the other recommended cutting it. As a result, the paper was reworked, and most of the credit for the linear time algorithm went to W. Dowling and J. Gallier, [19].

▶ Lesson 15. What may look as an exercise to you may still be an important result for others.

My answer to why Horn formulas matter in Computer Science may be found in [49], and my early advocating of the use of model theoretic methods in Computer Science in [55] and in [58].

Finite Model Theory

It took me a while to really grasp why the restriction to finite models in databases was such an important issue. In [54, 68] I tried to characterize database dependencies using preservation theorems. I knew from my previous work that the interpolation theorems of first-order logic fail when we look at finite models only. But I did not realize then, and had to learn it from Y. Gurevich that the classical preservation theorems also fail. A notable exception of these failures is B. Rossman's Homomorphism Preservation Theorem, [77]. It remains open, whether K. Compton's Preservation Theorem for classes of structures closed under disjoint union and taking of components, [12], has an analogue for finite structures.

I was aware of Fagin's characterization of **NP** using existential second-orderlogic, but only when M. Vardi and N. Immermann proved their characterization of **P**, I realized that they had actually proved some kind of Lindström Theorem in terms of Complexity Theory.

It took me a while to understand in depth that two ideas of early Model Theory, mostly neglected in logic monographs before 1985 with the exception of D. Monk's [71], would be pervasive in applications of Model Theory to Computer Science and Combinatorics: Ehrenfeucht-Fraïssé games and Feferman-Vaught-type Theorems, cf. [60].

My own dabblings in Finite Model Theory were first concerned, with moderate success, in explaining generalized quantifiers in terms of oracle computations and in trying to capture relativized complexity classes by using suitably chosen generalized quantifiers, cf. [66]. However, the use of oracles in low complexity classes depends subtly on the exact way oracles are accessed, cf. [7], and my treatment of the subject remained sketchy.

Monadic Second-Order Logic

Monadic Second-Order Logic over arbitrary structures is much stronger than First-Order Logic, and its semantics inherits problems of set theory. In contrast to this, over finite structures, Monadic Second-Order Logic seems natural and manageable. In 1995 Bruno

Courcelle visited the Technion and his visit was the beginning of an intense collaboration. B. Courcelle's book with J. Engelfriet, [13], gives a full account of the use of Monadic Second-Order Logic via a language theoretic approach. My own work in this direction started with Y. Pnueli, A. Pnueli's nephew, with whom I proved a hierarchy theorem for Second-Order Logic over finite structures: Let $AA_{m,n}$ be the class of properties of structures definable in Second-Order Logic with m alternations of second-order quantifiers using relation variables of arity at most n.

▶ **Theorem 5** (JAM and Y. Pnueli, [65]). *The hierarchy formed by $AA_{m,n}$ is strict.*

In this period I had three PhD students: U. Rotics, E. Ravve and G. Kogan, working with me in three different directions: With E. Ravve I tried to find applications of Feferman-Vaught-like theorems to system verification, a project which gave limited success due to complexity limitations, and the fact that we did not work out a real life example. U. Rotics, my former student in Databases, approached me, after working for several years in industry, with ideas on how to generalize tree-width of graphs. Finally, G. Kogan , a new immigrant from the former Soviet Union, came to my colleague M. Kaminski and me with ideas on how to compute permanents of special classes of matrices, see [41]. Unfortunately, he was not able to complete the necessary non-mathematical requirements and failed to turn his excellent work into an orderly PhD. With U. Rotics we discovered independently the notion of clique-width introduced by B. Courcelle, J. Engelfriet and G. Rozenberg in [14] and further developed in [18], which led to [15, 16, 17].

▶ **Theorem 6** (B. Courcelle, JAM and U. Rotics). *Let $CW(k)$ be the class of graphs G of clique-width at most k, and Φ denote a decision problem, optimization problem or counting problem, or even a graph polynomial, which is definable in Monadic Second-Order Logic. Then Φ can be solved on graphs in $CK(k)$ is polynomial time.*

When we proved this, we had to assume that the graph G was given together with its parse-tree witnessing its clique-width. However, this assumtion can be eliminated using results of R. Seymour and S. Oung [75, 74].

Graph Polynomials and Knot Theory

G. Kogan inspired me to apply the techniques developed in [17] to the computation of permanents. Let M be an $(n \times n)$-matrix over some field $\mathbb{F}$. M is orthogonal over $\mathbb{F}$ if $MM^t = I$. M has rank at most k over $\mathbb{F}$ if it has at most k rows (columns) which are linearily independent. M has tree-width at most k if the graph $G_M = ([n], E_M)$ has tree-width at most k, where $(i,j) \in E_M$ iff $m_{ij} \neq 0$.

▶ **Theorem 7.** *Let $\mathbb{F}$ be any field.*

A. Barvinok [4] *If M has rank at most k, $\mathrm{per}(M)$ can be computed in polynomial time, where the constants depend on k.*

JAM, 1997, cf. [17] *If M has tree-width at most k, $\mathrm{per}(M)$ can be computed in polynomial time, where the constants depend on k.*

G. Kogan, [41] *If $\mathbb{F}$ has characteristic 3, and M is orthogonal over $\mathbb{F}$, $\mathrm{per}(M)$ can be computed in polynomial time.*

I clearly felt that my result on permanents had little to do with permanents and I was looking for other applications of the techniques developed in [17]. While on sabbatical in Zurich, I met V. Turaev and told him about my result. He suggested I should try to apply

these techniques to the Jones polynomial in Knot Theory. I spent a year learning Knot Theory, especially knot polynomials and the Tutte polynomial which led to [59, 46, 61, 6].

A knot diagram is a planar signed graph. Its size is the number of crossings. If the knot diagram is alternating, it can be represented by an unsigned graph. The tree-width of signed graph is the same as the tree-width of the underlying unsigned graph. F. Jaeger, [40], showed that the Jones polynomial of an alternating knot diagram is essentially the Tutte polynomial.

▶ **Theorem 8.** *Let G be a graph and D be a knot diagram.*

A. Andrzejak [1], S. Noble [73] *If G is of tree-width at most k the Tutte polynomial can be computed in polynomial time and is FPT (fixed parameter tractable). The same holds for the Jones polynomial of alternating knots.*

JAM, [59, 61] *If D is a (not necessarily alternating) knot diagram of tree-width at most k the Jones polynomial can be computed in polynomial time and is FPT (fixed parameter tractable).*

This led me to study more graph polynomials with the ambitious goal of developing a general framework in which graph polynomials can be compared, cf. [63].

▶ Lesson 16. Do not restrict your supervising of PhD students to your own predefined topics.

Back to categoricity

In 2005 the CSL conference was held in Oxford. It was the first CSL conference after I was elected president of EACSL. B. Zilber, an old friend from the times I worked in Model Theory on the finite axiomatizability of categorical theories, was now professor of Mathematical Logic in Oxford. Using the occasion, I went to see B. Zilber. While I was explaining to him my work on graph polynomials, he noticed that my examples of graph polynomials occurred as size functions of finite approximations in totally categorical theories. I could not believe what he told me! Instead of attending the CSL lectures we started to explore this further, and indeed, it worked. We began to work out a general theory of graph polynomials using model theoretic methods which resulted in the papers [69, 42, 43].

My scientific trip which started in Model Theory went far afield, to Databases, Logic Programming, Algorithmics and Complexity, only to return me to my origins. Now I work in applications of Model Theory to Finite Combinatorics. In January 2009 M. Grohe and I have organized a special session, *Model Theoretic Methods in Finite Combinatorics*, at the Joint AMS-ASL meeting in Washington, D.C. The book [31] is the result of this special session.

Giving Credit

My narrative mentions several time the issue of giving credit to others. Clearly, it is not possible to remember precisely and all the time who or what inspired us to get our research results. Our memory is not reliable and conversations with colleagues which do not affect our work immediately tend to be forgotten. I have sinned on these accounts, and so have most of us. A bit of concerted introspection, however, helps a lot in avoiding careless omissions.

▶ Lesson 17. One cannot be careful enough in giving credit.

Logicians and Computer Scientists

Anthropologists study humans and their *cultural systems* consisting of people sharing a purpose and certain tools and values. Cultural systems are well defined objects of study

which were adapted by R.L. Wilder in [82] to the study of the evolution of mathematical concepts. Professional organisations, both of the logicians and the emerging community of computer scientists, had difficulties in acknowledging the relevance of Logic to Computer Science from the point of view of their respective disciplines.

On a personal level, when I was hired in 1980 by the Technion in Haifa, I was met with great suspicion, in spite of or because of the rôle logic played in Israel. The three founding fathers of Computer Science in Israel, M. Rabin, E. Shamir and S. Even, all understood the relevance of logic to Computer Science, but those coming from the culture of Electrical Engineering did not. The establishment of LiCS as an IEEE conference came as a complete surprise for them.

On a more global scale, in these early years neither the Association of Symbolic Logic, nor its German (rather German language) counterpart the DVMLG, showed genuine interest in the new partnership of Logic and Computer Science. Both the LiCS and CSL conferences were founded in 1986 and 1987 respectively because they had to create their own research community based on this partnership. In the last twenty or so years this partnership has thrived and spawned many new subcommunities, some of them concerned with foundational questions, but many using logical tools for genuine computer engineering disciplines much like calculus and differential equations are used in traditional engineering disciplines.

My own involvement in scientific organizations was within EACSL, LiCS and the German Logic Association (DVMLG). I served as vice-president and president of EACSL from 2002 until 2010. I also served on the board of DVMLG in the same period. I had myself several goals set to be realized during my terms:

- To strengthen the cooperation between LiCS and EACSL;
- To increase the visibility of EACSL;
- To increase the control of the scientists over their publication media;
- To further acceptance of Logic for Computer Science also as part of traditional Logic;
- To strengthen European activities of DVMLG.

I am happy to say that a good part of this agenda was realized. During my time

- the Ackermann Award was created;
- Cooperation between LiCS and EACSL was firmly established;
- The CSL proceedings moved finally from Springer to LIPIcs;
- The journal *Mathematical Logic Quarterly* became formally affiliated with DVMLG;
- Logic in Computer Science is now well represented on the board of DVMLG;
- Cooperation between EACSL and CiE (Computing in Europe) and KGS (Kurt Gödel Society) are well established;
- DVMLG held its first joint meeting with the Polish Logic Society in 2010.

Today the importance of logic for computer science seems to be well recognized and the Turing Awards of M. Rabin and D. Scott (Automata Theory), A. Pnueli (Temporal Logic) and E. Clark, A. Emerson and J. Sifakis (Model Checking) for the development of verification tools testify this. I said, "seems to be well recognized". The truth is that today many Computer Science Departments in which logic-based courses were a compulsory part of the curriculum tend to abandon teaching these courses. My suspicion is that this is partly due to the fact that we have not adapted the syllabi of our courses to the true needs of Computer Science. We tend to teach logic still in the tradition of the book by D. Hilbert and W. Ackermann [37] telling the same old stories about Hilbert's program, the paradoxes, and how K. Gödel put an end to the misguided hopes to use logic as the ultimate foundation of mathematics. I have detailed my thoughts about this in [50].

The standard course Linear Algebra evolved in the 1950s as an answer to physicists' needs providing them with the mathematical tools for quantum mechanics. We have yet to design a convincing logic course as an answer to the true needs of computer scientists and engineers. But we have to do this fast and in a concerted effort before it is too late and all our achievements are turned into well-used but ill-understood tools of the trade. I personally hope that the Turing Centenary will serve on an international scale as a reminder to the scientific public at large that we logicians still have something to offer to advance Computer (Computing) Science still further.

▶ Lesson 18. In your basic courses, show what one can do, and show its limitations, but do not speak mostly about the dashed hopes of the past.

Future Challenges

The scientific community at large, and we logicians in particular, face several challenges. Technological and economic changes lead to radical changes in research and teaching. Both are threatened by short-range commercial interests and the effects of mass production in education and research. Our traditional models of producing young scientists and engineers, and of producing and evaluating research do not scale. What proved itself over the centuries in small elitist communities fails to function at the current scale of scientific and technological activities. Knowledge used to be the source of enlightenment and emancipation. Therefore it was meant to be shared by large parts of mankind. Knowledge is the basis for being largely autonomous individuals. Today knowledge tends to be delegated to the CLOUD, and access to the CLOUD will be controlled by few. Education increasingly emphasizes the ability to merely use techniques rather than to understand them thoroughly. The slow disappearence of logic-based courses is only a symptom. Delegating knowledge to the CLOUD endangers our freedom and maturity.

> Enlightenment is man's emergence from his self-imposed immaturity. Immaturity is the inability to use one's understanding without guidance from another. This immaturity is self-imposed when its cause lies not in lack of understanding, but in lack of resolve and courage to use it without guidance from another. Sapere Aude! [dare to know] "Have courage to use your own understanding!"–that is the motto of enlightenment.
>
> E. Kant, "An Answer to the Question: What is Enlightenment?" (1784)

> I do not know whether we will ever reach mature adulthood. Many things in our experience convince us that the historical event of the Enlightenment did not make us mature adults, and we have not reached that stage yet. However, it seems to me that a meaning can be attributed to that critical interrogation on the present and on ourselves which Kant formulated by reflecting on the Enlightenment. It seems to me that Kant's reflection is even a way of philosophizing that has not been without its importance or effectiveness during the last two centuries. The critical ontology of ourselves has to be considered not, certainly, as a theory, a doctrine, nor even as a permanent body of knowledge that is accumulating; it has to be conceived as an attitude, an ethos, a philosophical life in which the critique of what we are is at one and the same time the historical analysis of the limits that are imposed on us and an experiment with the possibility of going beyond them.
>
> M. Foucault "What is Enlightenment ?" ("Qu'est-ce que les Lumières?"), in Rabinow (P.), ed., The Foucault Reader, Pantheon Books, 1984, pp. 32-50.

Being an "autonomous individual" an utopia, but striving to approximate being one is still a noble task.

> Even in teaching mathematics we can at least attempt to teach students the flavour of freedom and critical thought, and to get them used to the idea of being treated as humans empowered with the ability to understand.
>
> Roger Godement, Cours d'Algèbre, Hermann, Paris 1966 (my translation)

References

1 A. Andrzejak. An algorithm for the Tutte polynomials of graphs of bounded treewidth. *Discrete Mathematics*, 190:39–54, 1998.

2 I. Averbouch, B. Godlin, and J.A. Makowsky. An extension of the bivariate chromatic polynomial. *European Journal of Combinatorics*, 31.1:1–17, 2010.

3 J.T. Baldwin and A.H. Lachlan. On strongly minimal sets. *J. Symbolic Logic*, 36(1):79–96, 1971.

4 A.I. Barvinok. Two algorithmic results for the traveling salesman problem. *Mathematics of Operations Research*, 21:65–84, 1996.

5 J. Barwise and S. Feferman, editors. *Model-Theoretic Logics*. Perspectives in Mathematical Logic. Springer Verlag, 1985.

6 M. Bläser, H. Dell, and J.A. Makowsky. Algebraic point-to-point reductions for the colored Tutte polynomial. Preprint, 2007.

7 J.F. Buss. Alternations and space–bounded computations. *Journal of Computer and System Sciences*, 36:351–378, 1988.

8 A. Chandra, H. Lewis, and J.A. Makowsky. Embedded implicational dependencies and their implication problem. In *ACM Symposium on the Theory of Computing 1981*, pages 342–354. ACM, 1981.

9 A. K. Chandra, H. R. Lewis, and J. A. Makowsky. Embedded implicational dependencies and their inference problem. In *XP1 Workshop on Database Theory*, 1980.

10 C.C. Chang and H.J. Keisler. *Model Theory*. Studies in Logic, vol 73. North–Holland, 3rd edition, 1990.

11 G. Cherlin, L. Harrington, and A. Lachlan. $\aleph_0$-categorical $\aleph_0$-stable structures. *Annals of Pure and Applied Logic*, 18:227–270, 1980.

12 K. J. Compton. Some useful preservation theorems. *J. Symb. Log.*, 48(2):427–440, 1983.

13 B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-order Logic, a Language Theoretic Approach*. Cambridge University Press, 2012, in press.

14 B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. System Sci.*, 46:218–270, 1993.

15 B. Courcelle, J.A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graph of bounded clique width, extended abstract. In J. Hromkovic and O. Sykora, editors, *Graph Theoretic Concepts in Computer Science, 24th International Workshop, WG'98*, volume 1517 of *Lecture Notes in Computer Science*, pages 1–16. Springer Verlag, 1998.

16 B. Courcelle, J.A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33.2:125–150, 2000.

17 B. Courcelle, J.A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second order logic. *Discrete Applied Mathematics*, 108(1-2):23–52, 2001.

18 B. Courcelle and S. Olariu. Upper bounds to the clique–width of graphs. *Discrete Applied Mathematics*, 101:77–114, 2000.

19 W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. Log. Program.*, 1(3):267–284, 1984.

20 H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations und Initial Semantics*, volume 6 of *Monographs in Theoretical Computer Science. An EATCS Series*. Springer, 1985.

21 E. Engeler. A characterization of theories with isomorphic denumerable models. *Not. Amer. Math. Soc.*, 6:161, 1959. Abstract.

22 Erwin Engeler. *Untersuchungen zur Modelltheorie*. PhD thesis, Department of Mathematics, ETH-Zurich, Switzerland, 1958.

23 S. Feferman. Applications of many–sorted interpolation theorems. In L. Henkin et al., editor, *Proceedings of the Tarski Symposium*, volume 25 of *Proceedings of Symposia in Pure Mathematics*, pages 205–223. American Mathematical Society, 1974.

24 S. Feferman. Two notes on abstract model theory, I: Properties invariant on the range of definable relations between strcutures. *Fundamenta Mathematicae*, 82:153–165, 1974.

25 S. Feferman. Two notes on abstract model theory, II: Languages for which the set of valid sentences is s.i.i.d. *Fundamenta Mathematicae*, 89:153–165, 1975.

26 S. Feferman and R. Vaught. The first order properties of algebraic systems. *Fundamenta Mathematicae*, 47:57–103, 1959.

27 Z. Galil. On the complexity of regular resolution and the davis-putnam procedure. *Theor. Comput. Sci.*, 4(1):23–46, 1977.

28 B. Godlin, E. Katz, and J.A. Makowsky. Graph polynomials: From recursive definitions to subset expansion formulas. *Journal of Logic and Computation*, in press:xx–yy, 2011. DOI: 10.1093/logcom/exq006.

29 J. Goguen, J.W. Thatcher, E.G. Wagner, and J.B. Wright. Initial algebra semantics and continuous algebras. *Journal of ACM*, 24:68–95, 1977.

30 J. A. Goguen and R. M. Burstall. Introducing institutions. In *Logic of Programs*, pages 221–256, 1983.

31 M. Grohe and J.A. Makowsky, editors. *Model Theoretic Methods in Finite Combinatorics*, volume 558 of *Contemporary Mathematics*. American Mathematical Society, 2011. in press.

32 O. Grumberg, N. Francez, J.A. Makowsky, and W.P. de Roever. A proof rule for fair termination of guarded commands. *Information and Control*, 66.1-2:83–102, 1985.

33 Y. Gurevich. The word problem for some classes of semigroups (russian). *Algebra and Logic*, 5(2):25–35, 1966.

34 Y. Gurevich. Logic and the challenge of computer science. In E. Börger, editor, *Trends in Theoretical Computer Science*, Principles of Computer Science Series, chapter 1. Computer Science Press, 1988.

35 D. Harel. A general result on infinite trees and its applications (preliminary report). In *STOC*, pages 418–427, 1984.

36 D. Harel. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *J. ACM*, 33(1):224–248, 1986.

37 D. Hilbert and W. Ackermann. *Principles of Mathematical Logic*. Chelsea Publishing Company, 1950.

38 A. Itai and J.A. Makowsky. On the complexity of Herbrand's theorem. Technical Report No. 243, Departmnet of Computer Science, Technion–Israel Institute of Technology, Haifa, Israel, 1982.

39 A. Itai and J.A. Makowsky. Unification as a complexity measure for logic programming. *Journal of Logic Programming*, 4(2):105–117, 1987.

40 F. Jaeger. Tutte polynomials and link polynomials. *Proceedings of the American Mathematical Society*, 103:647–654, 1988.

41 G.P. Kogan. Computing the permanent over fields of characteristic 3: Where and why it becomes difficult. In *FOCS'96*, pages 108–114. IEEE, 1996.

42 T. Kotek, J.A. Makowsky, and B. Zilber. On counting generalized colorings. In *Computer Science Logic, CSL'08*, volume 5213 of *Lecture Notes in Computer Science*, page 339–353, 2008.

43 T. Kotek, J.A. Makowsky, and B. Zilber. On counting generalized colorings. In M. Grohe and J.A. Makowsky, editors, *Model Theoretic Methods in Finite Combinatorics*, volume 558 of *Contemporary Mathematics*, pages xx–yy. American Mathematical Society, 2011. in press.

44 G. Kreisel. Choice of infinitary languages by means of definability criteria; generalized recursion theory. In J. Barwise, editor, *The Syntax and Semantics of Infinitary Languages*, volume 72 of *Springer Lecture Notes in Mathematics*, pages 139–151. Springer, 1968.

45 D. J. Lehmann, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: The ethics of concurrent termination. In *ICALP*, pages 264–277, 1981.

46 M. Lotz and J.A. Makowsky. On the algebraic complexity of some families of coloured Tutte polynomials. *Advances in Applied Mathematics*, 32(1-2):327–349, 2004.

47 B. Mahr and J.A. Makowsky. Characterizing specification languages which admit initial semantics. *Theoretical Computer Science*, 31:49–60, 1984.

48 J. A. Makowsky. Measuring the expressive power of dynamic logics: An application of abstract model theory. In *ICALP*, pages 409–421, 1980.

49 J. A. Makowsky. Why Horn formulas matter for computer science: Initial structures and generic examples. *Journal of Computer and System Sciences*, 34.2/3:266–292, 1987.

50 J. A. Makowsky. From Hilbert's program to a logic tool box. *Ann. Math. Artif. Intell.*, 53(1-4):225–250, 2008.

51 J. A. Makowsky and I. Sain. Weak second order characterizations of various program verification systems. *Theoretical Computer Science*, 66:299–321, 1989.

52 J.A. Makowsky. Langages engendrés à partir des formules de Scott. *C.R. hebd. Acad. Sc. Paris*, 276:1585–1587, 1973.

53 J.A. Makowsky. On some conjectures connected with complete sentences. *Fund. Math.*, 81:193–202, 1974.

54 J.A. Makowsky. Characterizing database dependencies. In *ICALP'81*, volume 115 of *Lecture Notes in Computer Science*, pages 86–97. Springer Verlag, 1981.

55 J.A. Makowsky. Model theoretic issues in theoretical computer science, part I: Relational databases and abstract data types. In G. Lolli et al., editor, *Logic Colloquium '82*, Studies in Logic, pages 303–343. North Holland, 1984.

56 J.A. Makowsky. Abstract embedding relations. In J. Barwise and S. Feferman, editors, *Model-Theoretic Logics*, Perspectives in Mathematical Logic, chapter 20. Springer Verlag, 1985.

57 J.A. Makowsky. Compactness, embeddings and definability. In J. Barwise and S. Feferman, editors, *Model-Theoretic Logics*, Perspectives in Mathematical Logic, chapter 18. Springer Verlag, 1985.

58 J.A. Makowsky. Model theory and computer science: An appetizer. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, chapter I.6. Oxford University Press, 1992.

59 J.A. Makowsky. Colored Tutte polynomials and Kauffman brackets on graphs of bounded tree width. In *Proceedings of the 12th Symposium on Discrete Algorithms*, pages 487–495. SIAM, 2001.

60 J.A. Makowsky. Algorithmic uses of the Feferman-Vaught theorem. *Annals of Pure and Applied Logic*, 126.1-3:159–213, 2004.

61 J.A. Makowsky. Colored Tutte polynomials and Kauffman brackets on graphs of bounded tree width. *Disc. Appl. Math.*, 145(2):276–290, 2005.

62 J.A. Makowsky. Encounters with A. Mostowski. In W. Marek and M. Srebrny, editors, *70 years of Foundational Studies: In memoriam of Andrzej Mostowski*, page In press. IOS Press, 2008.

63 J.A. Makowsky. From a zoo to a zoology: Towards a general theory of graph polynomials. *Theory of Computing Systems*, 43:542–562, 2008.

64 J.A. Makowsky and D. Mundici. Abstract equivalence relations. In J. Barwise and S. Feferman, editors, *Model-Theoretic Logics*, Perspectives in Mathematical Logic, chapter 19. Springer Verlag, 1985.

65 J.A. Makowsky and Y. Pnueli. Arity vs. alternation in second order logic. *Annals of Pure and Applied Logic*, 78(2):189–202, 1996.

66 J.A. Makowsky and Y.B. Pnueli. Oracles and quantifiers. In *CSL'93*, volume 832 of *Lecture Notes in Computer Science*, pages 189–222. Springer, 1994.

67 J.A. Makowsky, S. Shelah, and J. Stavi. Δ–logics and generalized quantifiers. *Annals of Mathematical Logic*, 10:155–192, 1976.

68 J.A. Makowsky and M. Vardi. On the expressive power of data dependencies. *Acta Informatica*, 23.3:231–244, 1986.

69 J.A. Makowsky and B. Zilber. Polynomial invariants of graphs and totally categorical theories. MODNET Preprint No. 21, `http://www.logique.jussieu.fr/modnet/Publications/Preprint%20 server`, 2006.

70 V. M. Markowitz and J. A. Makowsky. Identifying extended entity-relationship object structures in relational schemas. *IEEE Trans. Software Eng.*, 16(8):777–790, 1990.

71 J.D. Monk. *Mathematical Logic*. Graduate Texts in Mathematics. Springer Verlag, 1976.

72 M. Morley. Categoricity in power. *Trans. Amer. Math. Soc.*, 114:514–38, 1965.

73 S. D. Noble. Evaluating the Tutte polynomial for graphs of bounded tree-width. *Combinatorics, Probability, and Computing*, 7:307–321, 1998.

74 S. Oum. Approximating rank-width and clique-width quickly. In *Graph Theoretic Concepts in Computer Science, WG 2005*, volume 3787 of *Lecture Notes in Computer Science*, pages 49–58, 2005.

75 S. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Ser. B*, 96(4):514–528, 2006.

76 M.G. Peretyat'kin. An example of an ω_1-categorical complete finitely axiomatizable theory. *Algebra and Logic*, 19:202–229, 1980.

77 B. Rossman. Homomorphism preservation theorems. *J. ACM*, 55(3), 2008.

78 D. Scott. Logic with denumerably long formulas and finite strings of quantifiers. In *Theory of Models*, Studies in Logic, pages 329–341, 1965.

79 J. Shoenfield. *Mathematical Logic*. Addison-Wesley Series in Logic. Addison-Wesley, 1967.

80 E. Specker and V. Strassen, editors. *Komplexität von Entscheidungsproblemen*, volume 43 of *Lecture Notes in Computer Science*. Springer Verlag, 1976.

81 L.H. Tharp. Continuity and elementary logic. *J. Symb. Log.*, 39(4):700–716, 1974.

82 R.L. Wilder. *Mathematics as a Cultural System*. Pergamon Press, 1981.

83 B. Zilber. *Uncountably Categorical Theories*, volume 117 of *Transl.of Math.Monographs*. Amercan Mathematical Society, Providence, R.I., 1993.